appviewx

SIGN+ Guides

Version: 2025.0.0.0

Copyright AppViewX, Inc.

Trademarks

Contact Information

AppViewX, Inc.

222 Broadway, FL 19

New York, NY 10038

Email: info@appviewx.com

Web: www.appviewx.com

Contents

Preface

Revision History

Revision	Description	Date
1.0	Initial draft of document for release 2025.0.0.0	November 2025

About this Guide

A comprehensive manual for executing Certificate Lifecycle Management (Enroll, Renew, Regenerate, and Revoke).

Audience

This guide is intended for all AppViewX Customers and Application Teams.

Third-Party Software Acknowledgments

This section serves as a placeholder to document the third-party components referenced in this guide, along with their associated trademark information.

Text Conventions

The following text conventions are used in this document:

Convention	Description
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`codeblock`	Indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Chapter 1: SIGN+ User Guide

Introduction

SIGN+ is an advanced software signing solution that enhances the security and trustworthiness of digital code and applications. It utilizes industry-standard practices to verify software's authenticity and integrity, offering strong protection against tampering and malware threats. This guide delves into SIGN+'s key features, benefits, and implementation across diverse software development and distribution scenarios.

Code Signing

Code signing is a security practice in software development and distribution. That involves digitally signing software executables and scripts to validate their authenticity and integrity, ensuring that the code remains unaltered and trustworthy since it was signed by the software publisher.

Code signing plays a vital role in ensuring software security and trustworthiness, serving multiple essential purposes:

1. **Authentication:** Code signing allows software developers and publishers to verify their identities. When developers sign their code, they use a digital certificate issued by a trusted certificate authority (CA). This certificate is associated with the developer's identity, signifying that the code can be trusted and originates from a known source.
2. **Integrity:** Code signing guarantees that the code remains unaltered or untampered with during transmission or after signing. The digital signature serves as a checksum for the code. Any modifications to the code render the signature invalid, indicating potential tampering.
3. **Trust:** End-users and systems can trust signed code. Operating systems and security software frequently examine the digital signature of code before permitting its execution. Valid signatures from trusted sources are more likely to be executed without warnings or restrictions.
4. **Security:** Code signing offers protection against malware and malicious tampering. If malicious actors attempt to alter a signed executable, the signature becomes invalid, and the modified code is not trusted.
5. **Version Control:** Code signing can include version information, enabling users to verify the software version during installation. This helps in managing software updates and patches.
6. **Timestamping:** Code signing certificates often include timestamp data. This ensures that the signature remains valid even after the certificate expires, which is crucial for long-lived software.
7. **Protection Against Malware and Tampering:** Code signing serves as a deterrent against distributing malware and tampering with code. Attackers are less likely to modify or inject malicious code into a signed application because such tampering would invalidate the digital signature and raise suspicion among users.
8. **Reducing False Positives:** Antivirus and security software frequently rely on code signing to assess the trustworthiness of applications. Signed software is less likely to trigger alerts as potential threats or false positives, resulting in smoother and safer user experiences.
9. **Smoother Installation and Execution:** Modern operating systems and security software often require signed code for installation and execution. Applications lacking signatures may trigger warning messages or encounter additional security prompts, potentially causing user hesitancy and inconvenience.

Enhancing Software Security Through Code Signing

Code signing enhances software security by:

1. **Verifying authenticity:** Users can trust the software's source and code origin.
2. **Ensuring integrity:** Tamper detection alerts against unauthorized changes.
3. **Malware Prevention:** Identifies malicious code injections.
4. **Secure software updates:** Ensures users receive authentic and untampered updates.
5. **Phishing Risk Reduction:** Digital signatures verify sender authenticity.
6. **Trust in Downloads:** Users trust verified software providers.
7. **Enterprise Security Enhancement:** Only approved and secure software is installed.
8. **Compliance Assurance:** Adheres to industry standards and regulations.

Key Components Involved in Code Signing

The code signing process involves several key components to ensure code authenticity and integrity. These components collaborate to establish trust between the code signer and the code recipient.

The key components involved in code signing are:

1. **Code Signing Certificate:** A digital certificate, issued by a trusted Certificate Authority (CA), specifically for code signing.
2. **Private Key:** Part of the code signing certificate used to create the digital signature during the signing process.
3. **Public Key:** Extracted from the code signing certificate, it verifies the digital signature and ensures the code hasn't been altered or tampered since signing.
4. **Digital Signature:** Created by hashing the code and encrypting the hash using the private key. It is embedded in the code to verify authenticity and integrity.
5. **Hash Function:** A cryptographic hash function is used to generate the code's hash for signing.
6. **Certificate Revocation Information:** This contains data about the code signing certificate's revocation and expiry status.
7. **Certificate Chain:** Comprises the code signing certificate and one or more intermediate certificates, ultimately leading to a trusted root certificate. It validates that the certificate is issued by a trusted CA.

Applications of Code Signing Across Different Software Types

- **Executable Files (.exe)**: These are applications and software programs.
- **Dynamic Link Libraries (.dll)**: Shared libraries used by applications.

- **Installer Packages (.msi)**: These are setup files for software installation.
- **Scripts**: This includes PowerShell scripts, VBScript, JavaScript, and more.
- **Java Archive Files (.jar)**: Used primarily in Java applications.
- **Mobile Apps**: This covers Android APK files and iOS app bundles.
- **Drivers**: Device drivers for various hardware components.
- **Plug-ins**: Extensions for software applications.
- **Firmware**: Embedded software found in hardware devices.

Overall, code signing is a critical practice for ensuring the authenticity, integrity, and security of software, making it a trusted component of software distribution and installation processes.

Code Signing Process

The code signing process involves the following steps:

1. **Obtaining a code signing certificate:** To sign the code, you need a code signing certificate from a trusted Certificate Authority (CA). This certificate will contain a public and private key pair, respectively, used for signing and verifying purposes.
2. **Hash Generation:** The next step in the code signing process is to generate a hash value for the software code to be signed. A hash function (such as SHA-256) is used to create a unique fixed-length string representing the code's content, known as the digest.
3. **Signing process:** The private key from the code signing certificate encrypts the generated hash, creating a unique digital signature that ensures the integrity and authenticity of the software.
4. **Embedding Signature:** The digital signature is subsequently embedded into the code or software. The method of embedding may vary depending on the platform and file format, ensuring it can be successfully verified.
5. **Verification:** When a user or system receives the signed code, the digital signature is extracted from the code. The digital signature is decrypted using the public key corresponding to the private key used to sign the code. This confirms that the signature matches the code's content and that the code has not been altered since signing, thus remaining untampered.
6. **Additional Verification:** Some additional verification is also carried out, such as the certificate chain validation for checking if the certificate comes from a trusted CA and the revocation check to ensure the signing certificate has not been revoked or expired. These are given as warnings and provide more trust and transparency.

Key Details in a Code Signing Certificate

The information included in a code signing certificate typically consists of the following:

1. **Subject Name:** This includes the name of the entity or individual specified on the certificate. It may be an individual's name or the name of a company or organization.
2. **Subject Alternative Names (SANs):** In some cases, the certificate may include multiple subject names (SANs), allowing the certificate to be used for signing code on different domains or platforms.
3. **Serial Number:** The issuing CA assigns a unique identifier to the certificate.
4. **Public Key:** The code signing certificate contains a public key that corresponds to the private key used for signing the code. The private key remains with the signer and is used to generate the digital signature.
5. **Issuer:** Information about the certificate authority that issued the code-signing certificate. This helps verify the certificate's authenticity.
6. **Validity Period:** The certificate's start and end dates define the period during which the certificate is considered valid for code signing.
7. **Thumbprint/Fingerprint:** A hash value calculated from the certificate's content, serving as a unique identifier for the certificate.
8. **Key Usage:** Indicates the purpose for which the public key can be used. For code-signing certificates, this would typically include the "Digital Signature" key usage.
9. **Extended Key Usage:** A list of specific purposes for which the certificate can be used. For code-signing certificates, this would include "Code Signing".
10. **Certificate Revocation Information:** Code-signing certificates are subject to revocation if they are compromised or invalidated. The certificate may contain information about how to check for revocation, such as Certificate Revocation Lists (CRLs) or Online Certificate Status Protocol (OCSP) responders.
11. **Signature Algorithm:** The algorithm used to sign the certificate itself, ensuring its integrity and authenticity.

Code Signing Certificate

A trusted certificate authority (CA) issues a code signing certificate, also referred to as a software signing certificate or digital code signing certificate. Software developers and publishers use this certificate to sign their software code, scripts, and executables. These certificates play a crucial role in ensuring the authenticity and integrity of software distributed to end-users.

• Need for a Code Signing Certificate

Need for a Code Signing Certificate

In code signing, digital certificates are central and crucial for establishing trust in the authenticity and integrity of the signed code. Digital certificates serve several key roles, including:

1. **Verification of Code Signer Identity:**
   - The digital certificate holds information regarding the identity of the code signer, including the organization's or individual's name, email address, and other relevant details.
   - Verifying the code signer's identity is vital for users to determine the legitimacy and trustworthiness of the code's source.
2. **Creation of Digital Signature:**
   - The code signer utilizes their private key, which is part of the code signing certificate, to generate a digital signature for the code.
   - The signature is unique to the code and the signer, that helps establish the authenticity of the code.
3. **Verification of Code Integrity:**
   - Users or systems receiving the signed code use the public key from the code signing certificate to decrypt the digital signature.
   - This process generates the original hash value of the code, which is compared with a newly calculated hash of the received code.
   - If the two hash values match, that indicates that the code has not been altered or tampered with since signing, ensuring the code's integrity.
4. **Certificate Chain Verification:**
   - A trusted Certificate Authority issues the digital certificate, and the certificate chain verifies its authenticity.
   - This step establishes trust in the code signer's identity, as the CA is responsible for validating the identity before issuing the certificate.
5. **Revocation Checking:**
   - Digital certificates can be revoked if they are compromised, expired, or no longer considered trustworthy.
   - Revocation checking helps ensure that the code signing certificate is still valid and trustworthy at the time of code execution.
6. **Identity Verification:**
   - To acquire a code signing certificate, a software developer or publisher must undergo a thorough identity verification process conducted by the certificate authority (CA).
   - This process verifies the identity and legitimacy of the developer. Once verified, the CA issues a digital certificate associated with the developer's name or organization.
7. **Digital Signature:**
   - With a code signing certificate in hand, developers can digitally sign their software code and files. This involves applying a unique cryptographic signature to the code, which serves as a tamper-evident seal.
   - Any modification to the code after signing will invalidate the signature.
8. **Authenticity:**

- When end-users or systems encounter signed software, they can verify the digital signature using the public key associated with the code signing certificate.
- This verification process confirms that the software has not been altered since it was signed and that it indeed comes from the verified developer or publisher. It helps users trust the software's authenticity.

9. **User Trust:**
   - Code signing certificates are essential for establishing trust between software developers and end-users.
   - When users download and run signed software, they are less likely to encounter security warnings or alerts, as the signature signifies that the software is from a reputable source.
10. **Protection Against Malware:**
    - Code signing is a preventive measure against malware and malicious tampering. Signed software is less likely to be modified by attackers, as any changes would break the signature.
    - This makes it more challenging for malicious actors to distribute compromised software.
11. **Version Control:**
    - Code signing certificates can include version information, allowing users to verify the specific version of the software they are installing. This is particularly important for software updates and patches.

In summary, the need for a code signing certificate arises from the critical role it plays in establishing trust, verifying the authenticity of software, and ensuring its integrity. By digitally signing their code, developers provide users with confidence that the software has not been tampered with and that it originates from a legitimate source. This is especially important in a digital landscape where security and trust are paramount.

Signing Policy

A signing policy, in the context of code signing and security practices, refers to a set of rules, guidelines, and procedures that govern how digital signatures are applied to software, scripts, or other digital assets.

Signing policies are typically defined and implemented within organizations to ensure the secure and consistent application of digital signatures. These policies are a fundamental part of a broader security strategy and are important for various reasons:

- **Security Assurance:** Signing policies help ensure the security of software and digital assets by specifying who can sign code, what can be signed, and under what circumstances. They establish a framework for mitigating risks associated with unauthorized or malicious code modifications.
- **Authentication:** Signing policies often dictate the use of code signing certificates issued by trusted certificate authorities (CAs). These certificates verify the identity of the signer, adding a layer of authentication to the signed code. This helps establish trust in the source of the software.
- **Integrity:** Policies define the conditions under which code should be signed. By adhering to these policies, organizations maintain the integrity of their code base, as any unauthorized changes or tampering will result in the invalidation of the digital signature.
- **Non-Repudiation:** Code signing with adherence to policies provides non-repudiation, meaning that the signer cannot deny their involvement in the signing process. This is crucial for accountability and legal purposes.
- **Compliance:** Many industries and regulatory bodies require organizations to adhere to specific code signing practices. Signing policies helps ensure compliance with these regulations, which is especially important in sectors like healthcare, finance, and government.
- **Version Control:** Policies can specify how versioning should be managed for signed code. This helps users verify the authenticity and integrity of software updates and patches.

Key Aspects Covered by Signing Policies

A signing policy plays a crucial role in an organization's cybersecurity strategy by fostering trust, preserving code integrity, and mitigating the risk of malware and security breaches. It provides explicit guidelines for secure code signing practices, making it an essential component of secure software development and distribution. Key aspects of security addressed by signing policies include:

- **Authorized Signers:** Signing policies are used to determine authorized personnel, identifying individuals within the organization authorized to sign code or digital assets, which may include specific developers or security team members
- **Signing Environment:** Secure code signing environments identify and include environments and systems that are secure and trusted.
- **Certificate Usage:** Managing code signing certificates addresses the selection and management of the certificates, often emphasizing the use of certificates issued by recognized Certificate Authorities (CAs).
- **Review and Approval:** Code review and approval procedures ensure compliance with security and quality standards before signing.
- **Time-stamping:** Signing policies ensure valid signatures over time, implementing timestamping requirements to maintain the validity of signatures, even after the certificate's expiration.
- **Revocation:** Signing policies outline procedures for revoking signatures in cases of compromised certificates or unauthorized code changes.

TimeStamping

TimeStamping is a cryptographic technique used to securely record the exact date and time when a digital document or code was signed. A trusted timestamp authority (TSA), typically operated by a certificate authority (CA) or a third-party timestamping service, generates this timestamp. It ensures long-term validity of digital signatures by providing an immutable reference point, preventing disputes about the timing of the signature's creation.

• Timestamping in Code Signing

Timestamping in Code Signing

Timestamping in code signing involves adding a timestamp to the digital signature of a software application or code package. The timestamp is cryptographically incorporated into the digital signature itself, thus ensuring the long-term validity and trustworthiness of the code signature, even after the signing certificate has expired.

When code is signed, a digital signature is generated using the code signer's private key. This signature is based on the code's content and includes metadata about the signer and the signing certificate. However, digital certificates have a finite validity period, and they can be revoked if compromised or no longer considered trustworthy. Once a certificate expires or is revoked, the signature becomes invalid.

Benefits of timestamping:

- **Long-Term Validity:** By including a timestamp in the code signature, the signature remains valid even after the code signer's certificate has expired or been revoked. The timestamp establishes the signing time while the certificate is still valid.
- **Non-Repudiation:** The timestamp serves as proof that the code was signed at a specific time by a specific entity. This helps prevent the signer from denying their involvement in the signing process, providing non-repudiation of the signature.
- **Trust Across Time:** End-users can trust the signature, knowing that it was valid at the time of signing, even if the signing certificate is no longer valid. This is especially important for the long-term archival of code or when verifying the authenticity of older, signed code.
- **Protection Against Time-Based Attacks:** Including a timestamp helps protect against potential attacks aimed at exploiting vulnerabilities in code signatures that depend solely on certificate validity periods.

- **Certificate Expiration:** Digital certificates used for code signing have a limited lifespan, typically ranging from one to three years. When a certificate expires, any code signed with that certificate may be considered invalid, leading to potential security issues and software functionality problems.
- **Trustworthiness:** Timestamps are issued by trusted timestamp authorities, adding an additional layer of trust to the code signature.
- **Security Updates:** Users can be confident that software updates or patches signed with an expired certificate are still valid if they have a valid timestamp.

Timestamping Process

The timestamping process for code signing includes the following steps:

1. The code signer signs the code using their code signing certificate.
2. The signer sends the code signature to a TSA for timestamping.
3. The TSA generates a timestamp token, which includes the UTC time and date.
4. The TSA's certificate signs the timestamp token.
5. The timestamp token is added to the code's digital signature.
6. The signed code, including the timestamp token, is distributed to users.

Timestamping Authorities Supported by AppViewX for Code Signing

- GlobalSign
- Symantec (now part of DigiCert)
- Entrust
- SwissSign
- Comodo CA (now Sectigo)
- DigiCert
- IdenTrust
- QuoVadis Global
- GlobalSign Advanced.

If you need to use a timestamping authority other than those listed above, you should provide the specific URL or information related to that timestamping authority. This ensures that your code signing process is properly configured to use the required timestamping service. Be sure to consult with your organization's code signing policies and requirements when selecting a timestamping authority or specifying a custom URL.

Getting Started

Know your Deployments for SIGN+

We support multiple deployment options to cater to various customer needs and infrastructure preferences. Our solutions can be deployed in On-Premises environments, Managed Kubernetes platforms such as EKS, AKS, GKE and OpenShift and as a Software as a Service (SaaS).

SaaS Deployment (Highly Secure and Hassle-Free)

Our Software as a Service (SaaS) offering is designed for organizations that prioritize security, simplicity, and efficiency. In this deployment mode, we manage all aspects of application hosting, maintenance, and scaling, providing a worry-free experience for our customers. Our SaaS platform is built with cutting-edge security measures, including robust encryption, multi-factor authentication, and continuous monitoring to ensure your data and operations are protected at all times.

Choosing SaaS not only reduces the burden on your IT teams but also ensures that you benefit from the latest updates, features, and security enhancements without any additional effort. This option is ideal for organizations of all sizes, particularly those looking to quickly access our services with the assurance of enterprise-grade security and compliance. For additional information about saas deployment, click here.

On-Premises Deployment

On-Premises deployment enables organizations to install and operate our applications on their own infrastructure. This approach offers the highest level of control and customization, making it particularly suitable for organizations with strict security, compliance, or performance needs. It is best suited for enterprises with dedicated IT resources and the expertise to manage complex infrastructure. For additional information about on-premises deployment, click here.

Managed Kubernetes Deployment (AKS, EKS, and GKE)

We offer support for deployment on Managed Kubernetes services such as Amazon EKS, Microsoft AKS, and Google GKE. Managed Kubernetes platforms simplify the management of Kubernetes clusters by handling tasks like cluster provisioning, maintenance, and scaling.

This deployment option is suitable for organizations looking for a balance between control and operational simplicity, and for those who prefer leveraging cloud providers' infrastructure and expertise. For additional information about managed kubernetes deployment, click here.

Overview

About AppViewX

AppViewX is advanced cybersecurity and network management, automation, and orchestration platform for Enterprise IT. AppViewX Lifecycle Management Solution for Certificates on ADC or Load Balancers, Servers, Firewall, Cloud, Web Application Firewall (WAF), and enterprise mobility solution aims to avoid network outages due to unplanned certificate expiration and improve organization security posture. This remote monitoring and management platform helps network operations move faster, enforce compliance, eliminate errors, and reduce costs in the organization.

SIGN+ Overview

AppViewX's SIGN+ provides an end–to–end lifecycle management of x.509 digital certificates across complex networks to secure your business. With SIGN+, security teams can manage the certificate lifecycle from an intuitive single-pane management interface. It enables the Certificate Lifecycle Management and Automation solution which helps enterprise IT manage and automate the entire lifecycle of their internal and external PKI. The key stages of the certificate lifecycle can be broken into the following stages:

Certificate Discovery and Inventory Management - This allows users to discover certificates across the network and manage inventory of all certificates in one place.

Visibility and Monitoring - This enables the user to monitor certificate expiry and usage. The monitored data is represented as a detailed report on the web portal along with options to trigger email alerts. Allows users to gain insights into certificates; monitor and take remedial action.

Certificate Enrollment - This allows users to request certificates from a certificate authority (CA) that confirms their identity and generates a certificate.

Certificate Renewal - This allows users to either manually or automatically renew a certificate before the expiry date by retaining the old private key.

Certificate Regeneration - This allows users to enroll new certificates with similar parameters to an old certificate. When a user generates a new private key, the user can modify the parameters if required.

Certificate Reissuance - This allows users to enroll new certificates with similar parameters to an old certificate. But the newly issued certificate comes with the same validity as the older certificate and can modify the parameters.

Certificate Revocation - This allows users to revoke a certificate in the event of certificate loss, compromise, or any other reason when the certificate is no more necessary for business.

Certificate Audit - Track and audit the usage, creation, expiration, and revocation of certificates. Track user interaction with the platform.

What is Certificate Lifecycle Management (CLM)?

There is a growing need for organizations to allow and control only specific individuals, devices, and machines to gain access to the network. The need for digital certificates to authenticate, identify, and control who can access and operate on an organization’s network. Managing digital certificates across complex networks to ensure protection and prevent failures is a must for all businesses. CLM ensures continuous monitoring of digital certificates, with the ability to audit and keep track of expirations and renewals to avoid any service disruption. The digital certificate is a mechanism by which machines and individuals are identified and authenticated.

What is x.509 Digital Certificate?

The digital certificate is a mechanism by which machines and individuals are identified and authenticated. Digital certificates (x.509 certificates) are essential to establish trust and authenticate the identity of machines, people, and so on.

It helps to verify the identity between users in operation, servers, and other entities in a network. Also, identifies servers from whom the encrypted data is received, the signer of information, and helps to establish authenticity and integrity. The x.509 digital certificate protects information belonging to enterprises and their customers.

A digital certificate contains:

- Name of the certificate holder.
- Serial Number that is used to uniquely identify the service, individual, or entity identified by the certificate.
- Expiry date.
- Copy of the certificate holder's public key (used for decrypting messages and digital signatures).
- Digital Signature of the certificate-issuing authority.

Certificate Authority

A Certificate Authority (CA) is also known as a certification authority or certificate issuer and is an establishment that validates the identities of certificate requestors and associates them to a cryptographic key through the issuance of electronic documents known as digital certificates.

Prerequisites

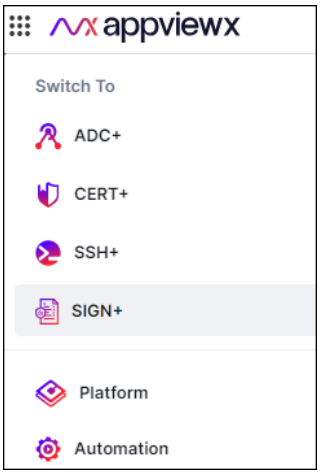
Supported Web Browsers

Web Browser	Version
Firefox	118.0.1 (64-bit) or later
Google Chrome	117.0.5938.134 (Official Build) (64-bit) or later

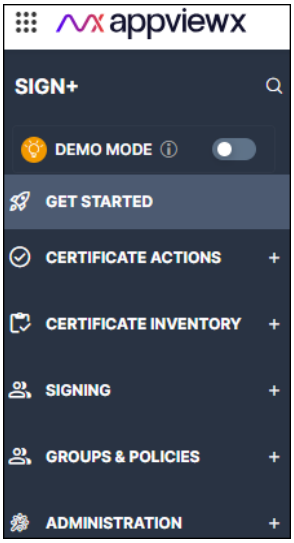
Accessing SIGN+

The following steps explain how to access **SIGN+**:

1. Log into AppViewX with valid credentials. (The product URL will be shared by AppViewX in the onboarding emails).
2. From the upper left corner of the screen, click  (**Menu**).
3. On the displayed menu click **SIGN+**.



The SIGN+ home page is displayed. The menu is replaced with the menu for **SIGN+**.



SIGN+ Onboarding

This guide outlines a step-by-step process for initiating customer onboarding onto the AppViewX SIGN + platform and getting started with the **AppViewX SIGN+** product. You can initiate SIGN+ in just a few simple steps, ensuring to safeguard the authenticity and integrity of your code.

Prerequisites

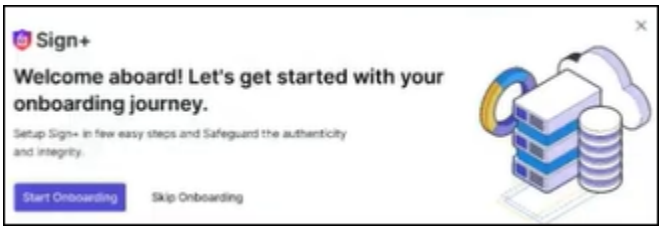
- Make sure you have the code signing certificate before starting with the onboarding journey.
- Make sure there are no policies configured before starting the onboarding journey.

Start with your Onboarding journey

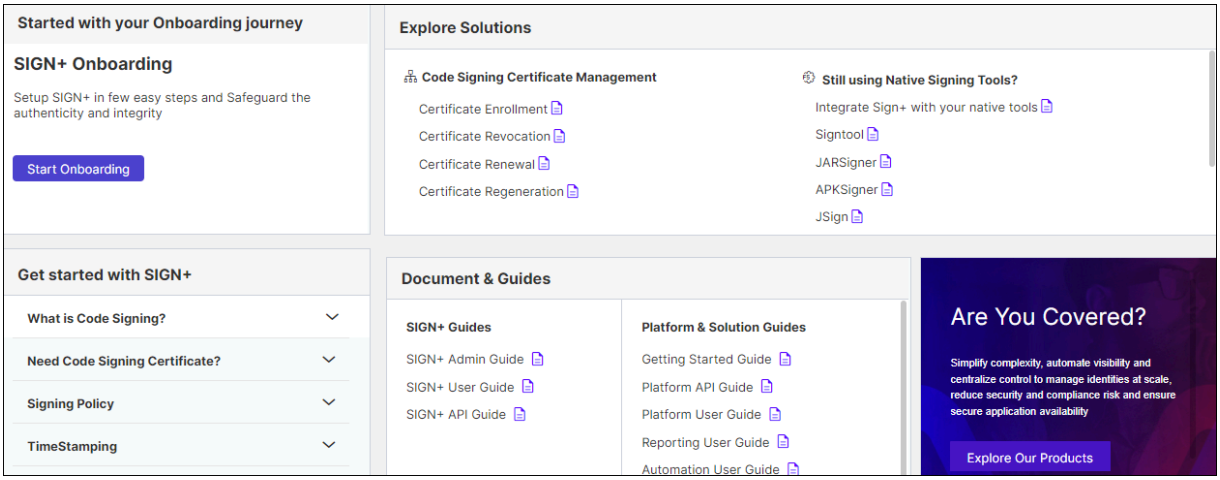
To initiate the onboarding process, follow these steps:

1. Login to AppViewX using your valid credentials.
2. If the policy is not configured, the **Sign+** popup is displayed.

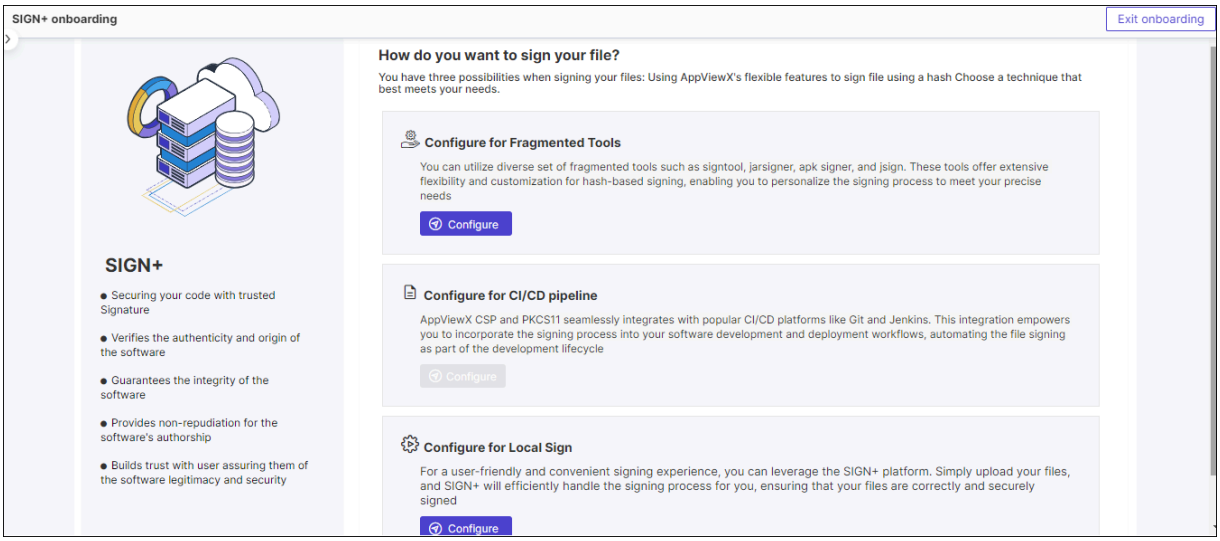
a. **Sign+** popup has the **Start Onboarding** button.



b. If you have configured the policy, you can locate the **Start Onboarding** button on the **Getting Started** page.



3. Click **Start Onboarding** to begin the onboarding process.
   The **SIGN+ Onboarding** page is displayed.



4. Select the method that aligns with your code signing requirements and click **Configure**.
   - **Configure for Fragmented Tools** **(Hash-based signing policy)** You can utilize a diverse set of fragmented tools such as signtool, jarsigner, apk signer, and jsign. These tools offer extensive flexibility and customization for hash-based signing, enabling you to personalize the signing process to meet your precise needs.
   - **Configure for CI/CD pipeline** AppViewX CSP and PKCS11 seamlessly integrate with popular CI/CD platforms like Git and Jenkins. This integration empowers you to incorporate the signing process into your software development and deployment workflows, automating the file signing as part of the development lifecycle.
   - **Configure for Local Sign** **(File based signing policy)** For a user-friendly and convenient signing experience, you can leverage the SIGN+ platform. Simply upload your files, and SIGN+ will efficiently handle the signing process for you, ensuring that your files are correctly and securely signed.

• Create Hash Based Signing Policy
• Create File Based Signing Policy

Create Hash Based Signing Policy

1. To Configure the **Signing Policy**.
   Enter the required policy details:

Fields	Description
***Policy Name**	Enter a unique name for the signing policy. No special characters other than **'.', '-','_'** are allowed. The name should not start with special characters.
***Hash Function**	Select the hash function you want to configure for code signing: [Dropdown Options - **SHA-1**, **SHA-256**, **SHA-384**, **SHA-512**]
***Timestamping**	Choose a trusted timestamping authority from the dropdown list: [Dropdown Options - **DigiCert, Entrust, Global Sign, IdenTrust, Sectigo, Other, None**]. If you choose **Other**, kindly provide the **timestamping URL**. **Note:** If you select **None**, the **Timestamping** will not be applied to the configured signing policy.
***Signing Type**	By default this is set to **Hash Based**.
***Restriction Type**	Select **None** or between **IP-based restriction** or **IP range-based restriction**.
***Number Of Polls**	Add the number of polls if the certificate is based on HSM, and Specify the total number of polls to be conducted within the designated polling interval and the value must be an integer between 5 and 20.
***Polling Interval**	Add the Polling Interval if the certificate is based on HSM, Set the time interval between consecutive polls and the value must be an integer between 10 and 300000 milliseconds.
***List of IP's**	**Note:** This field is displayed when the **Restriction Type** is set as **IP**. If you selected **IP-based restriction**, enter a list of valid individual IP addresses at subnet or system level.
***Start IP** ***End IP**	**Note:** This field is displayed when the **Restriction Type** is set as **IP Range**.

Fields	Description
	If you selected an **IP range-based restriction**, enter the start and end IP addresses, ensuring the end IP is greater than the start IP.
Test Policy	Enable the toggle to create the policy for internal testing. Enabling this option ignores all signatures associated with the policy in the license counting.
Enable Email notification	Enable the toggle button to receive email notifications and updates via email when the signing events occur.
*: *Mandatory fields*	

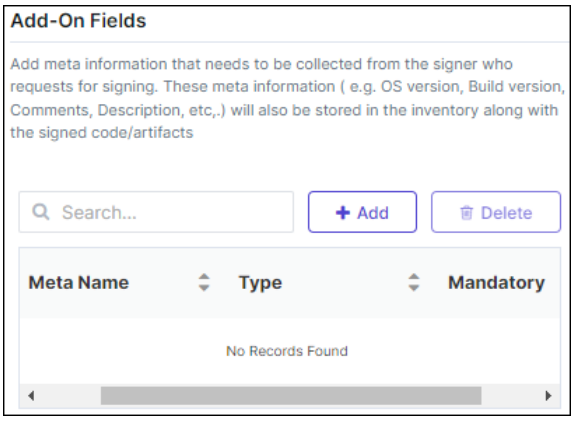
2. *(Optional step)* If the **Enable Email notification** toggle switch is enabled then enter the **Email Configuration** details as follows.

Fields	Description
***Email Subject**	Enter the subject line for the email notification to identify the purpose or content of the email. Acceptable characters are letters, numbers, and spaces.
***To**	Enter one or more recipients' email addresses separated by comma.
***Event Type**	Choose the type of events for which notifications are required. The values are **Success**, **Failure**, or **Both**.
***Required Field**	A multi-select dropdown field with values - **Policy name**, **Signing Type**, **Key Name**, **IP Address**, **Signing Time**, and **Username**. Select one or more values whose details are to be displayed in the mail body for comprehensive notification.
*: *Mandatory fields*	

3. In the **Map Signing Key** section, select the required signing keys from the dropdown.

> **Note:** If one or more signing keys are mapped to a policy then the signing key should be chosen as an option in the Upload & Sign or the default signing key will be used for signing.

4. In the **Add-On Fields** section, add meta information that needs to be collected from the signer who requests for signing.



a. To add metadata Click **+ Add**.

The **Add Data** page is displayed.

b. Configure metadata using following fields

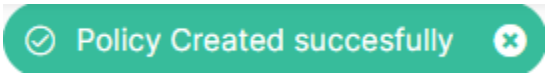
- **Meta Name:** Enter a unique name for a meta information.
- **Type:** Select a valid field type for validating the meta information field.
- **Mandatory:** Enable the toggle to make meta information a mandatory field while code signing.

c. Click **Add**.

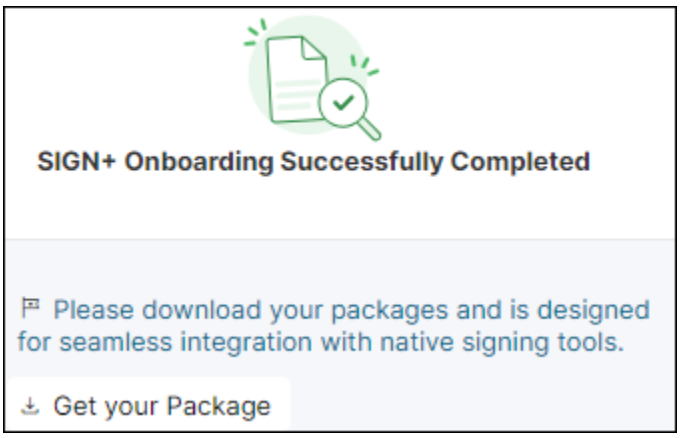
The **Add-On Fields** will be added in the meta information table.

5. Click **Create**.

The **Policy Created Successfully** message is displayed and policy is added in the inventory.



6. If the **SIGN+ Onboarding is Successfully Completed**.



7. Click **Get your Package**.

The **Download Package** page is displayed.

8. Configure the **Download Package**.

• Configure the Download Package

Configure the Download Package

To Download the pre-configured Sign+ package tailored for your selected operating system, policy, key, and user. This package is designed for seamless integration with native fragmented signing tools.

1. In the **General Details** section, select the values as follows:

Fields	Description
***OS Type**	Select the operating system of your choice from - **Windows**, **Linux**, or **Mac**.
***Authentication Type**	Select the type of authentication from • **User-Based:** User-based authentication verifies identity through user name credential. • **OAuth-Based:** Authentication through OAuth-based authorization through service account.
***User Name**	**Note:** This field is displayed if **Authentication Type** is selected as **OAuth-Based**. Select the username from the dropdown for which the SIGN+ package installer is required.
***Service Account**	**Note:** This field is displayed if **Authentication Type** is selected as **OAuth-Based**. Select the service account from the dropdown for which the SIGN+ package installer is required. For creating a service account in appviewx, click here.
*: *Mandatory fields*	

2. In the **Signing Configuration Details** section, select the values as follows:

a. To add a Signing configuration, Click **+ Add**.

The **Add Data** page is displayed.

Fields	Description
***Select Signing Policy**	Select the Signing policy name which is specified to a user group. **Note:** The default policy name is set as the first policy in the dropdown.
***Select Signing Key**	Select the sign key using which the uploaded file can be signed. **Note:** The default signing key is set as the signing key of the first policy in the dropdown.
*: *Mandatory fields*	

b. Click **Add**.

The selected Policy and Signing Key are displayed in a table.

The contents of the table are described below.

Column	Description
Policy Name	The policy selected in the field **Select Signing Policy**.
Key Name	The policy selected in the field **Select Signing Key**.

3. To delete a Policy from the table select the policy and click **Delete**.
4. To download the package files, click **Download**.

Create File Based Signing Policy

1. To configure **Signing Policy**.

Enter the required policy details:

Fields	Description
***Policy Name**	Enter a unique name for the signing policy. No special characters other than **'.', '-','_'** are allowed. The name should not start with special characters.
***Hash Function**	Select the hash function you want to configure for code signing: [Dropdown Options - **SHA-1**, **SHA-256**, **SHA-384**, **SHA-512**]
***Timestamping**	Choose a trusted timestamping authority from the dropdown list: [Dropdown Options - **DigiCert, Entrust, Global Sign, IdenTrust, Sectigo, Other, None**]. If you choose **Other**, kindly provide the **timestamping URL**. **Note:** If you select **None**, the **Timestamping** will not be applied to the configured signing policy.
***Signing Type**	By default this is set to **File Based**.
***File Types**	**Note:** This field is displayed only when the **Signing Type** is set as **File Based**. Select one or more file types that should be signed using the signing policy. Supported file types include **PS1, EXE, CAT, MSI, JS, JAR, APK, VBS, CAB, WSF, DLL, PSM1, PSD1, PS1XML, JSE, and VBE** among others. **Note:** Selected file types will only be permitted for upload and signing under this policy. **Note:** Signing operations for the HSM-based certificates for the script files will be supported by upgrading the JSign Version from 3.0 to 6.0. **Restriction:** CAT files do not work with HSM-based certificates, but work for File Based certificates.

Fields	Description
***Restriction Type**	Select **None** or between **IP-based restriction** or **IP range-based restriction**.
***Number Of Polls**	Add the number of polls if the certificate is based on HSM, and Specify the total number of polls to be conducted within the designated polling interval and the value must be an integer between 5 and 20.
***Polling Interval**	Add the Polling Interval if the certificate is based on HSM, Set the time interval between consecutive polls and the value must be an integer between 10 and 300000 milliseconds.
***List of IP's**	**Note:** This field is displayed when the **Restriction Type** is set as **IP**. If you selected **IP-based restriction**, enter a list of valid individual IP addresses at subnet or system level.
***Start IP** ***End IP**	**Note:** This field is displayed when the **Restriction Type** is set as **IP Range**. If you selected an **IP range-based restriction**, enter the start and end IP addresses, ensuring the end IP is greater than the start IP.
Test Policy	Enable the toggle to create the policy for internal testing. Enabling this option ignores all signatures associated with the policy in the license counting.
Enable Email notification	Enable the toggle button to receive email notifications and updates via email when the signing events occur.
*: *Mandatory fields*	

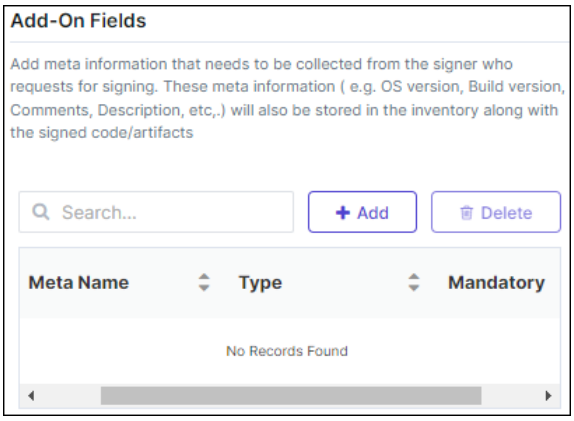
2. *(Optional step)* If the **Enable Email notification** toggle switch is enabled then enter the **Email Configuration** details as follows.

Fields	Description
***Email Subject**	Enter the subject line for the email notification to identify the purpose or content of the email. Acceptable characters are letters, numbers, and spaces.
***To**	Enter one or more recipients' email addresses separated by comma.
***Event Type**	Choose the type of events for which notifications are required. The values are **Success**, **Failure**, or **Both**.
***Required Field**	A multi-select dropdown field with values - **Policy name**, **Signing Type**, **Key Name**, **IP Address**, **Signing Time**, and **Username**. Select one or more values whose details are to be displayed in the mail body for comprehensive notification.
*: *Mandatory fields*	

3. In the **Map Signing Key** section, select the required signing keys from the dropdown.

> **Note:** If one or more signing keys are mapped to a policy then the signing key should be chosen as an option in the Upload & Sign or the default signing key will be used for signing.

4. In the **Add-On Fields** section, add meta information that needs to be collected from the signer who requests for signing.



   a. To add metadata Click **+ Add**.
   The **Add Data** page is displayed.

   b. Configure metadata using following fields

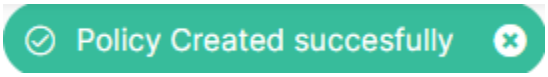
- **Meta Name:** Enter a unique name for a meta information.
- **Type:** Select a valid field type for validating the meta information field.
- **Mandatory:** Enable the toggle to make meta information a mandatory field while code signing.

c. Click **Add**.

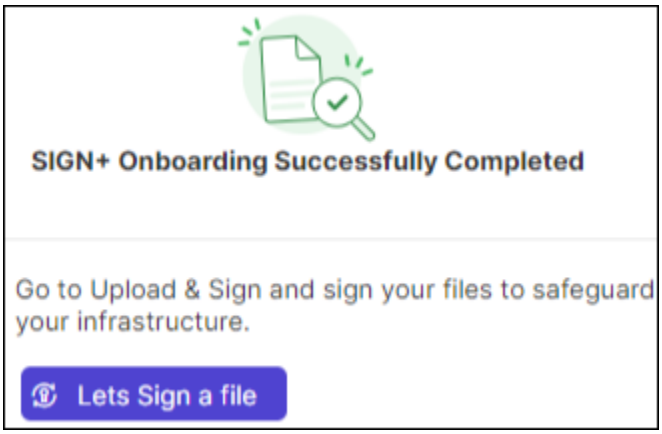
The **Add-On Fields** will be added in the meta information table.

5. Click **Create**.

The **Policy Created Successfully** message is displayed and policy is added to the signing inventory.



6. If the **SIGN+ Onboarding is Successfully Completed**.



7. Click **Lets Sign a file**.

The **Upload and sign** page is displayed.

8. Configure the **Upload and sign**.

• Upload and Sign

Upload and Sign

The **Upload and Sign** page enables users to upload a file for digital signing. On this page, users can choose created signing policies and the associated Signing keys to sign their uploaded files. Some signing policies may include specific meta-information configurations, ensuring that the signed files adhere to predefined policy requirements.

Upload and Sign a file using Code Signing Certificate, follow these steps:

1. Enter the required details, under the **General Details** section.

Fields	Description
***Select Signing Policy**	Select the suitable signing policy according to your specific needs, considering any configurations applied during the signing policy's creation. **Note:** The default policy name is set as the first policy in the dropdown.
***Select Signing Key**	Select the signing key that corresponds to the policy from the dropdown. **Note:** The default signing key is set as the signing key of the first policy in the dropdown.
***Upload File**	Upload the code signing file. Only selected file types during policy creation are allowed for upload.
*: *Mandatory fields*	

2. Click **Sign** to initiate the signing process.
   File Uploaded Successfully for Signing message is displayed after successful completion of the signing process.

   The **Signing Inventory** page is displayed.

What to do next:

Download Code Signed files that have been digitally signed and verified.

Downloading the CSP/PKCS#11 Package

1. Go to **(Menu)** > **SIGN+** > **Getting Started** (left menu).
   The Getting Started page is displayed.
2. In the **Downloads** widget (AppViewX Native Library) > , click **Download your CSP / PKCS#11 package**.
   The **Pages > Download Package** page is displayed.

3. In the **General Details** section select the values as follows:

General Section - Field Description table

Fields	Description
***OS Type**	Select the operating system of your choice from - **Windows**, **Linux**, or **Mac**.
***Authentication Type**	Select the type of authentication from • **User-Based**: User-based authentication verifies identity through user name credential. • **OAuth-Based**: Authentication through OAuth-based authorization through service account.
***User Name**	**Note:** This field is displayed if **Authentication Type** is selected as **User-Based**. Select the username from the dropdown for which the SIGN+ package installer is required.
***Service Account**	**Note:** This field is displayed if **Authentication Type** is selected as **OAuth-Based**. Select the service account from the dropdown for which the SIGN+ package installer is required. For creating a service account in appviewx, click here.
***Connection Type**	Select the option to download the CSP/PKCS#11 package through either Compute Cluster or Cloud Connector.

Fields	Description
	Note: a. You can download the CSP/PKCS11 package through the compute cluster only if no cloud connectors are configured in the cluster. b. This option applies only to non-HSM-based certificates. For HSM-based certificates, configuring a cloud connector is mandatory to ensure proper communication with the HSM. c. The Custom Connector URL feature in SIGN+ enhances flexibility by allowing communication with the SIGN + server via a load balancer. This setup enables the configuration of multiple dedicated Cloud Connectors, ensuring efficient routing of user requests from the end user's machine to the appropriate connector.
***CC Host Name**	**Note:** This field is displayed if **Connection Type** is selected as **Cloud Connector**. Select the cloud connector host name from the dropdown.
*: *Mandatory fields*	

4. In the **Signing Configuration Details** section select the values as follows:

Signing Configuration Details - Field Description table

Fields	Description
***Select Signing Policy**	Select the Signing policy name which is specified to a user group. **Note:** The default policy name is set as the first policy in the dropdown.
***Select Signing Key**	Select the sign key using which the uploaded file can be signed.

Fields	Description
	Note: The default signing key is set as the signing key of first policy in the dropdown.
*: *Mandatory fields*	

5. Click **Add**.
   The selected Policy and Signing Key are displayed in a table. The contents of the table are described below.

   Column description of Signing Configuration Details

Column	Description
Policy Name	The policy selected in the field **Select Signing Policy**.
Key Name	The policy selected in the field **Select Signing Key**.
Action	Contains a delete icon to remove the value from the table.

   Note: Once the Policy is added, it cannot be added a second time.

6. To download the package files, click the **Download** button.
   The package files are saved at the file location on your system. Different types of files are generated based on the selected **OS Type**. The files and their hierarchy are as follows:
   - **Windows**
     a. **Sign+ Installer file**
     b. **<Policy_Name>** folder - contains the root, intermediate, and code-signing certificates.
     c. **Libraries** - contains the folders **CSP**, **Dependencies**, and **PKCS11** along with **avx_sign_config.json**.
   - **Linux**
     a. **Sign+ Installer file**
     b. **<Policy_Name>** folder - contains the root, intermediate, and code-signing certificates.
     c. **Libraries** - contains the folder **PKCS11** along with **avx_sign_config.json**.
   - **Mac**
     a. **Sign+ Installer file**
     b. **<Policy_Name>** folder - contains the root, intermediate, and code-signing certificates.
     c. **Libraries** - contains the folder **PKCS11** along with **avx_sign_config.json**.

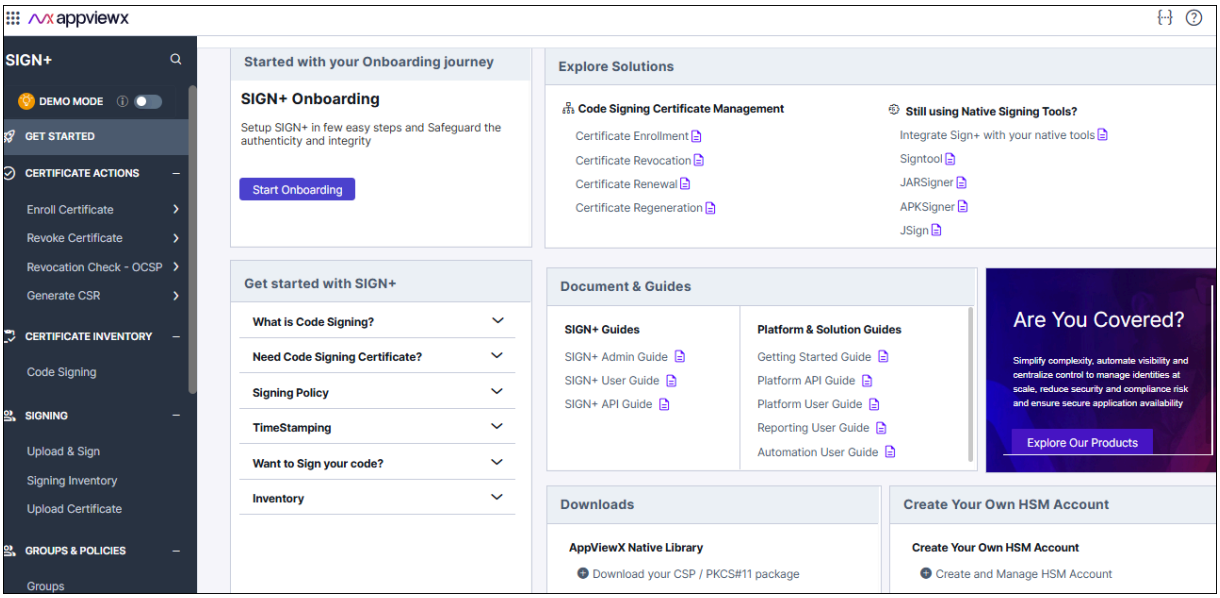
Code Signing as a Service

The Code Signing as a Service feature in AppViewX SIGN+ streamlines the onboarding experience for new customers by allowing the creation of a new tenant within the application. It enables users to quickly and easily create and configure a Hardware Security Module (HSM) account. By offering a seamless and efficient setup, it makes it easier for customers to begin using AppViewX SIGN+ for their code signing.

Note: For Code Signing as a Service, we currently support only **Fortanix HSM**.

Note: Ensure the HSM configuration is stored. If not, please contact AppViewX support.

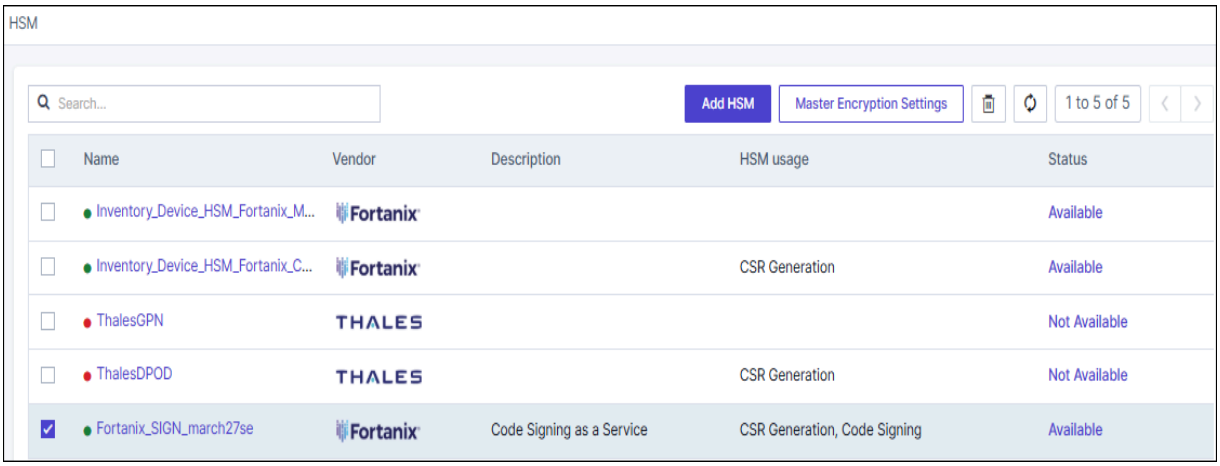
1. Go to **(Menu)** > **SIGN+** > **GETTING STARTED**.
2. On the **GETTING STARTED** page, locate the **Create Your Own HSM Account** section, then click **Create and Manage HSM Account**.



   The **Create HSM Account** page is displayed.
3. Select the required **Data Center** from the dropdown list where you wish to create your HSM account.
4. Click **Create HSM Account**.

   A confirmation message will appear indicating that your HSM account has been successfully created.

   You will be automatically redirected to the **HSM Inventory** page, where the newly created HSM account will be listed.



Note: To know more details on configuring Fortanix HSM, click here.

Certificate Actions

For launching any new application in an enterprise, an SSL certificate is required to secure the communication. The certificate lifecycle has different phases starting with enrolling. AppViewX SIGN+ enables you to manage every action that is involved in the certificate lifecycle.

In the Certificate Action section, the following actions can be performed:

- Enroll Certificate
- Revoke Certificate
- Revocation Check - OCSP
- Generate CSR
- Certificate Enrollment
- Certificate Revocation
- Revocation Check for Code Signing Certificate
- Generating CSR for Code Signing Certificate

Certificate Enrollment

Code signing certificate enrollment is the process of obtaining a digital certificate (from the Certificate Authority (CA)) that is specifically designed for signing code, scripts, executables, and software applications. This certificate is essential for software developers and organizations to verify the

authenticity and integrity of their software. It is a primary step in certificate lifecycle management (CLM). In the enrollment process, a user must submit the details of the entity (server or individual) to the certifying authority. The authority validates the correctness of the information and ownership before issuing a digital certificate.

Code Signing Certificate Enrollment

Code Signing certificate enrollment refers to the process of creating a digital ID for a code or document. It starts with the generation of a key pair (private and public key) and CSR and then submitting the CSR to the desired CA to procure a certificate. SIGN+ supports the generation of key pairs on the device, HSM, and AppViewX. You can also upload the CSR when enrolling for a digital certificate.

Note: These certificates cannot be hosted on servers.

Prerequisites

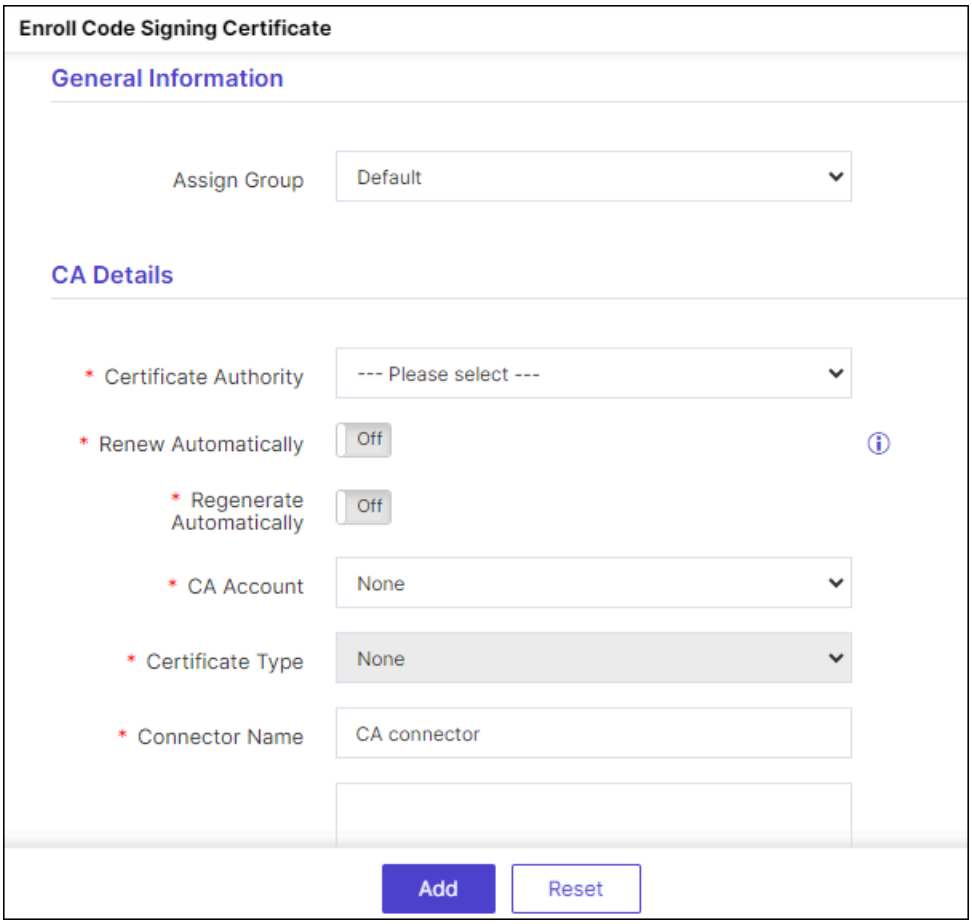
- Users should have read and write access to the account.
- The user should have configured the CA account in AppViewX.
- Policy creation and certificate profile are created according to the customer's use case.
- Purpose and usage are mapped according to the extended key usage and validation policy.

Enrollment

The following steps explain how to enroll a code signing certificate:

1. Go to **(Menu)** > **SIGN+**.
2. Under the **CERTIFICATE ACTIONS**, select **Enroll Certificate > Code Signing Certificate**.

   The **Enroll Code Signing Certificate** page is displayed.



3. In the **General Information** section, from the dropdown list, select the required **Assign Group**.
4. Enter the following fields in the **CA Details** section:

Field descriptions for the CA Details section

Fields	Description
***Certificate Authority**	Select the desired certificate authority from the dropdown lists. Based on the selected CA, other CA details are configured. The possible CAs are: • Digicert MPKI • GlobalSign SSL • GlobalSign MSSL • Microsoft Enterprise • Microsoft Standalone • Nexus

Fields	Description
	• OpenTrust • Any Other Programmable CA configured by the user
***Renew Automatically**	Select the toggle button to On or Off. • When the toggle is enabled, the Start Renewing option will be enabled. • Enter the number of days to renew the certificate automatically. **Note:** Changing the group inherited renew period overwrites the renewal period for this certificate.
***CA Account**	To which account the enrollment request to be submitted.
Certificate Type	Select the desired certificate type from the dropdown list.
***Division**	Select the division to which the certificate must be enrolled. **Note:** This field will be shown only for Digicert CA.
Certificate Profile	Select the Profile to which the Certificate must enroll. **Note:** This field is applicable only for AppViewX CA and Google CA.
***Issuer Location**	Select the location of the issuer CA from the dropdown list. **Note:** This is applicable only for Google CA.
***Issuer Name**	Select the name of the issuer CA from the dropdown list. **Note:** This is applicable only for Google CA.
***Connector Name**	Enter the friendly name for Certificate Authority connector in this field which will be displayed in the holistic view on saving this form.
Description	Enter the description in this field.

<table>
<tr><th>Fields</th><th>Description</th></tr>
<tr><td></td><td>Note: You can enter a maximum of 2000 words in the field.</td></tr>
<tr><td>*CSR Generation</td><td>Select the CSR generation option as required.

Options are:
• AppViewX CSR Generation.
• UploadCSR - Uploaded CSR will be taken as a source to populate CSR parameters and submit to CA.
• Click the Browse button, and then the file.
• Click the Upload button to upload the selected file.
• On uploading CSR successfully, CSR parameters are automatically filled in the CSR section.
• HSM - Private key and CSR will be created in the selected HSM device based on CSR parameters given.

<table>
<tr><th>Fields</th><th>Description</th></tr>
<tr><td>*Device Type</td><td>Select the type of device as required. The possible options are:
• HSM Devices
• ADC Devices</td></tr>
<tr><td>*Vendors</td><td>Select the desired vendors from the dropdown list.
The possible vendors when device selected as HSM Devices:
• Fortanix
• PKCS11

The possible vendors when device selected as ADC Devices:
• Safenet
• Thales
• Fortanix</td></tr>
<tr><td>*Devices</td><td>Select the desired device from the dropdown list.</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Fields</th><th>Description</th></tr>
<tr><td></td><td>
<table>
<tr><th>Fields</th><th>Description</th></tr>
<tr><td></td><td>Note:
• By default, the None Selected option is enabled.
• When Device Type = ADC - User chooses from the list based on the vendors field selection.</td></tr>
<tr><td>*Key Handler Name</td><td>Enter the desired handler name in the field.</td></tr>
<tr><td>*Key Reference Name</td><td>Enter the Key Reference Name.
Note: This field appears only when Device Type = ADC Devices.</td></tr>
</table>
• End Point - Private key and CSR will be created in the selected End Point device based on CSR parameters given.
<table>
<tr><th>Fields</th><th>Description</th></tr>
<tr><td>Category</td><td>Select the desired category from the dropdown list. The possible options are:
• ADC
• Server
• Firewall</td></tr>
<tr><td>Vendor</td><td>Select the desired vendor from the dropdown list. The possible options are:
• AVI
• Citrix
• F5
• Ngnix Plus
• HAProxy</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Fields</th><th>Description</th></tr>
<tr><td></td><td>
<table>
<tr><th>Fields</th><th>Description</th></tr>
<tr><td></td><td>Note: Vendor list is populated based on the category, select the desired vendor from the dropdown list.</td></tr>
<tr><td>*Devices</td><td>Select the desired device from the dropdown list.
Note: By default, the None option is selected.</td></tr>
<tr><td>Tenant</td><td>Enter the tenant ID in this field.
Note: This field appears when you select category as ADC.</td></tr>
<tr><td>*CSR file name</td><td>Enter the name of the CSR file in this field.
Note: This field appears when you select category as Server.</td></tr>
<tr><td>*Partition</td><td>Enter the partition in this field.
Note: This field appears when you select category as Firewall.</td></tr>
<tr><td>*Key File Name</td><td>Enter the name of the key file in this field.</td></tr>
</table>
Note: For all CA types except Amazon, you have the option to generate the CSR.
• AppViewX - Private key and CSR will be created in AppViewX based on CSR parameters given.
</td></tr>
</table>

Fields	Description
*: *Mandatory fields*	

While enrolling certificates with policies using Google CA, the following points must be considered:

- **Certificate Enrollment - Strict Policy**
  - The Common Name will not be pre-filled from the policy.
  - The following validation appears based on strict policy guidelines.
    - If the Common Name's domain name is not present in the **Allowed Domain Name** list, an error validation will be shown upon saving the policy details.
- **Certificate Enrollment - Suggestive Policy**
  - The Common Name will not be pre-filled from the policy
  - The following validation will be seen based on strict policy guidelines.
    - If the Common Name's domain name is not present in the **Allowed Domain Name** list, the non-compliant policy will be created.
    - If the Common Name's domain name is present in the **Blocked Domain Name** list, an error validation will be shown upon saving the policy details.

5. Only for the EJBCA CA, enter the **Vendor Specific Details**.

Field descriptions for the Vendor Specific Details section.

Fields	Description
End entity user name	Enter the name of the end entity.
*** End Entity Profile Name**	Select the profile name from the dropdown list .
*** User Common Name**	Select the common name from the dropdown list.
*** Certificate Profile Name**	Select the certificate profile name from the dropdown list.
*: *Mandatory fields*	

6. Enter the following fields in the **CSR Parameters**.

Fields	Description
***Common Name**	The common name is one of the key values of Certificate Signing Request (CSR) to be present in the certificate. For example, <appviewx>. No special characters allowed except en dash (_) and hyphen (-).

Fields	Description
Subject Alternative Name	You can see the count of subject alternative names (SAN) available for a certificate in the CSR parameter section, inventory grid, and CA connector page. Select the subject alternative subject name from the dropdown list. The possible options are, • Select all • DNS • IP Address. **Note:** • Multiple values must be separated by a comma. • The cumulative count SANs appears in the certificate property pop-up window from the holistic view.
***Organization**	The organization name is one of the CSR parameters to be present in the certificate. This field will be auto-filled and editable based on the configuration in the selected group's policy.
Organization Unit	Organization Unit name is one of the CSR parameters to be present in the certificate. This field will be auto-filled and editable based on the configuration in the selected group's policy.
Locality	The locality name is one of the CSR parameters to be present in the certificate. This field will be auto-filled and editable based on the configuration in the selected group's policy.
State	The state name is one of the CSR parameters to be present in the certificate. This field will be auto-filled and editable based on the configuration in the selected group's policy.
***Country**	Country name is one of the CSR parameters to be present in the certificate. This field will be auto-filled and editable based on configuration. It must be a 2-letter country code (for example, US, and so on).
Email Address	The email contact details of the person responsible for maintaining the certificate. Enter the valid e-mail address.

Fields	Description
***Validity**	Enter the number in this field and select the entered validity list to be in Days, Months, and Years from the dropdown list.
Challenge Password	Challenge password is one of the CSR parameters to be present in the certificate. Password must contain at least one alphabet (uppercase and lowercase), one number, and one special character.
Confirm Password	Re-enter the same password to confirm that is entered in the Challenge Password field.
***Hash Function**	The Hash function with which the CSR has to be signed. Any information specific to any CA or vendor has to be covered in the Note section. This field is auto-filled and editable based on the configuration in the selected group's policy. **Note:** For **Certificate Authority** = **HydrantID**, irrespective of the hash function selected, by default, the CA returns a certificate with SHA256. Therefore, admins must restrict users from creating a certificate with a hash function other than SHA256. To accomplish this, create policy with a single hash value (SHA256).
***Key Type**	The key type is used while creating a private and public key pair. This field will be auto-filled and editable based on the configuration in the selected group's policy. The supported key types are **RSA**, **ECC** and **DSA**.
***Bit Length**	The bit length is used while creating a private and public key pair. This field will be auto-filled and editable based on the configuration in the selected group's policy.
*: *Mandatory fields*	

7. In the **Attachments** section, upload any additional documents that are relevant to the enrollment of the certificate (for example, approval emails).

Field descriptions for the Attachments section

Fields	Description
Name	Enter the alternate name for the document to be uploaded.
Comments	Enter the comments in this field. **Note:** You can enter a maximum of 2000 words in the field.
Upload File	Click the Upload button to select the file.

8. Other than the CSR fields, you can add organization-specific values along with CSR. These values will not be part of the certificate but will be available in the AppViewX inventory. For example, cost center. Inventory can be filtered based on these attributes as well. In the Certificate Attributes can be added under Administration > certificate attributes, it will be reflected on the enrollment page:
9. Enter the relevant details in the **Generic Fields**. These are default fields for maintaining the IP address and device information, if required.

Field descriptions for the Generic Fields

Fields	Description
Device Name	Enter the name of the device.
Application IP Address	Enter the IP address of the application.

10. In the **Vendor-Specific Details** section, enter the CA-specific details. Some of the CAs will expect additional details other than CSR parameters for their operational purposes.
    - By default, the **Certificate ID** is auto-populated based on the value entered in the **Common Name** field (in the **CSR Parameters** section).
    - The **Certificate ID** can be modified by the user.
    - If the user edits the **Certificate ID**, any change to the **Common Name** will not reflect in the **Certificate ID**.
    - If the user deletes the **Certificate ID**, the value of the **Certificate ID** field is set to the **Common Name** suffixed with the timestamp.
11. Click **Add**.

Once the details are added, you will be redirected to a page where the CSR and CA details are added as a connector. This page is called the holistic view and from here, any action on the certificate can be performed including provisioning the certificate to a server.

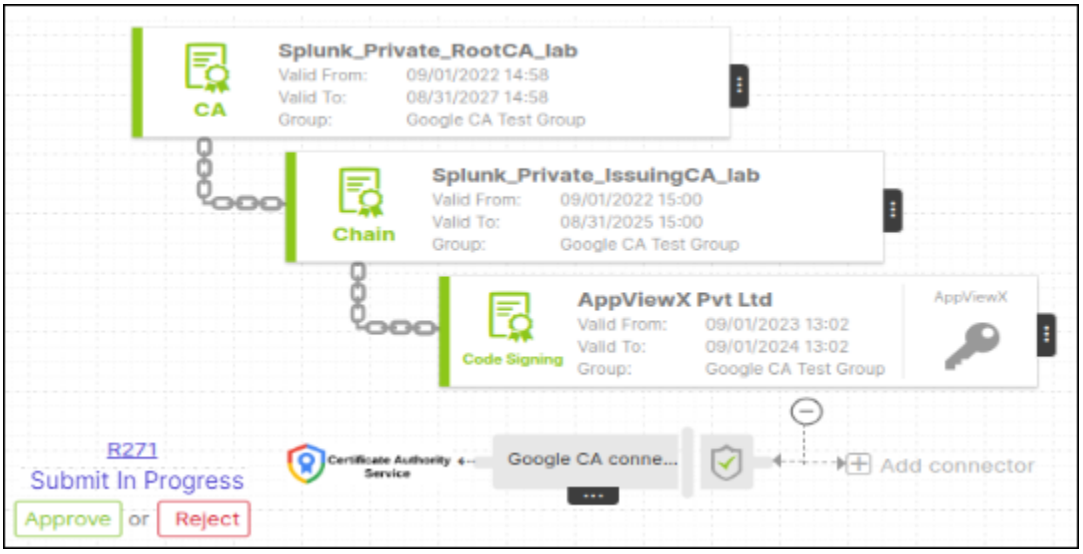
12. On the holistic view, click the **Submit** button to trigger the request.

    The submit action is triggered and the **Submit** dialog box is displayed.

13. Enter your comments in the text field and click **Yes**.

    If the approval required option is enabled in the CA policy, the request is moved to the **Approve** and **Implementation** stages.

14. Click **Approve** to proceed.



    The **Approve** dialog box is displayed.

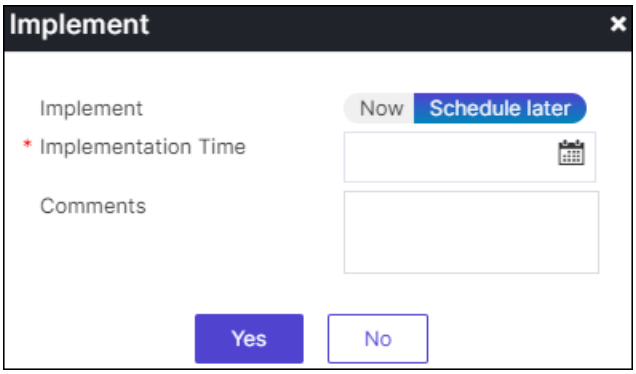
15. Enter your comments in the text field.

> **Note:** If the workflow request has to be approved automatically in the future, click the **Schedule later** button .

16. Click **Yes**.

    Once the approval process is completed, the **Implement** option is displayed in the holistic view.

17. On the certificate holistic view, click **Implement** to proceed.
18. In the **Implement** dialog box, enter your comments.



If the workflow request has to be implemented automatically in the future, click **Schedule later**. You can then select the **Implementation Time** from the calendar field.

19. Click **Yes**.

    CSR Submission to CA is in Progress.

20. Once the CSR submission is successful, the request state will be changed to **Submit** certificate - retrieval in progress state.

    If the enrollment request is compliant with conditions defined and auto-approval enabled in the targeted CA, the certificate will be fetched in a few seconds.

    If auto-approval disabled in the targeted CA, you will have to be logged into the CA and approve the request.

    Once the certificate is issued successfully, the certificate will be retrieved into AppViewX.

What to do next:

- Configure the signing policy with relevant details, ensuring mapping to the enrolled certificate (also identified as the signing key on the signing policy page).
- The file types selected during policy creation are the only ones permitted for upload. Supported file types include: PS1, EXE, CAT, MSI, JS, JAR, APK, VBS, CAB, WSF, DLL, PSM1, PSD1, PS1XML, JSE, and VBE.

Certificate Revocation

Revocation is the process of making a certificate invalid. For example, you might need to revoke a certificate if the certificate is no longer required or the certificate's private key is compromised. Make sure that you have permission to revoke a certificate and submit a request to the certificate authority. As soon as the certificate is revoked, it is not considered to be a trusted certificate. Revoked certificates are listed in the Certificate Revocation List (CRL) maintained by each certificate authority.

- Revoking Code Signing Certificate

Revoking Code Signing Certificate

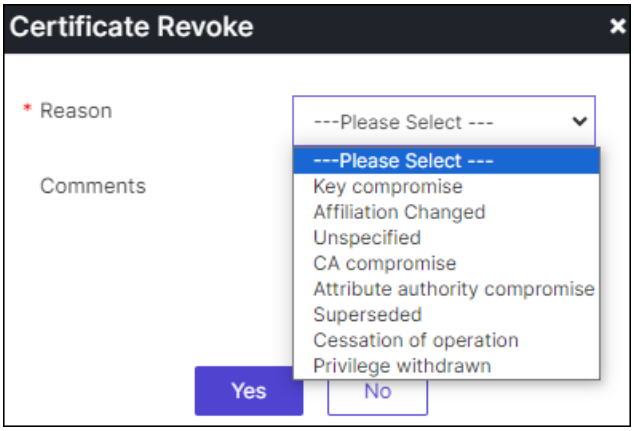
The following steps explain how to revoke Code Signing Certificate,

1. Go to **(Menu)** > **SIGN+**.
2. Under the **CERTIFICATE ACTIONS**, select **Revoke Certificate > Code Signing Certificate**.

   The **Code Signing Certificate** page is displayed.
3. To revoke a **Code Signing Certificate**, select the certificate name under **Common Name**.

   The holistic view of the selected certificate is displayed.
4. Hover over the three-dot menu for the certificate, and then click **Revoke**.

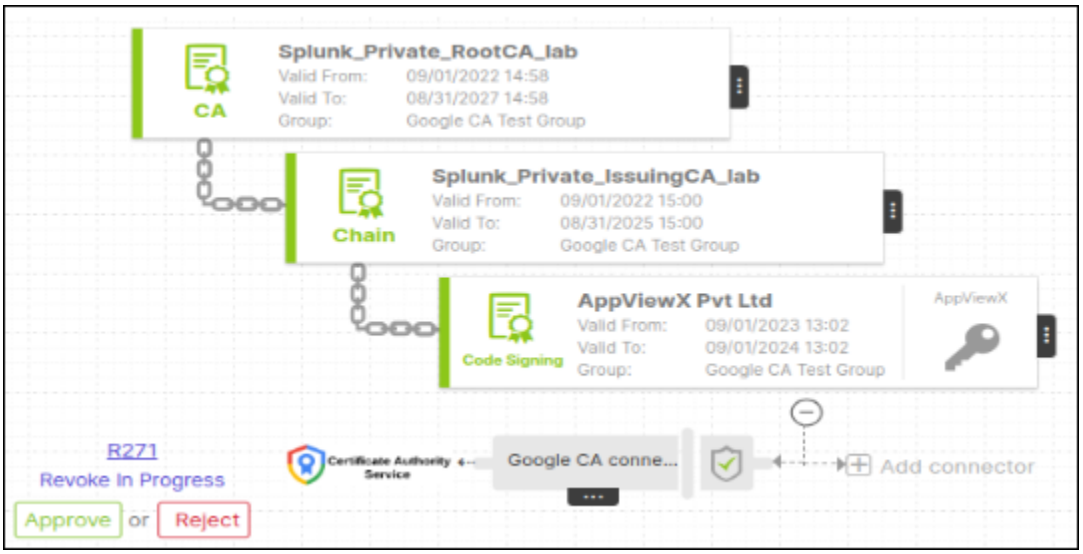
5. The **Certificate Revoke** dialog box is displayed.
6. In the **Certificate Revoke** dialog box, select the reason for revoking the certificate from the dropdown.
7. You can also enter comments in the Comments text field (optional), and then click **Yes**.



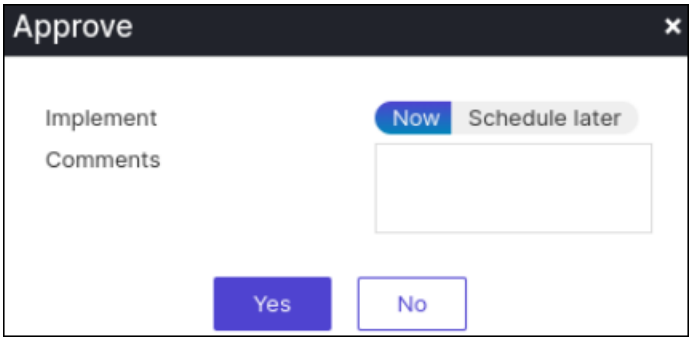
8. The revoke process is initiated for the selected certificate.

   If the **Approval Required** checkbox is enabled, the request will progress through the **Approve** and **Implementation** stages.

9. To proceed, click **Approve** on the certificate's holistic view.

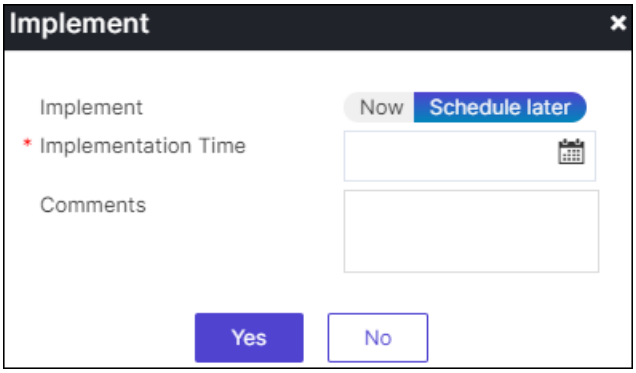


10. In the **Approve** dialog box, enter your comments (optional).



If you want to schedule automatic approval for the workflow request in the future, click **Schedule later**. You can then choose the Implementation Time from the calendar field.

11. Click **Yes**.
12. On the certificate holistic view, click **Implement** to proceed.
13. In the **Implement** dialog box, enter your comments.



If you want the workflow request to be automatically approved in the future, click **Schedule later**. You can then select the **Implementation Time** from the calendar field.

14. Click **Yes**.
15. After the certificate is revoked, the status updates to **Completed**.

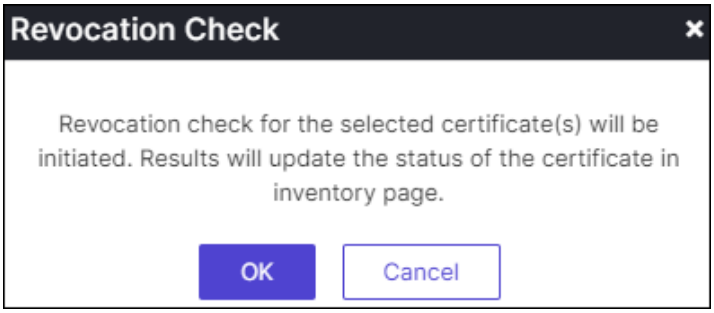
Revocation Check for Code Signing Certificate

The following steps explain how to perform revocation check for a code signing certificate:

1. Go to **(Menu)** > **SIGN+**.
2. Under the **CERTIFICATE ACTIONS**, select **Revocation Check - OCSP > Code Signing Certificate**.

   The **Code Signing Certificate** page is displayed.
3. To Revocation Check a **Code Signing Certificate**, select the certificate for which you want to perform a **Revocation Check** under **Common Name**.
4. Click Actions menu and choose **Revocation Check** from the dropdown.

   In the **Revocation Check** dialog box is displayed.



5. Click **OK** to proceed revocation check.
6. Revocation check results will be updated on the inventory page.

Generating CSR for Code Signing Certificate

The following steps explain how to generate a CSR for a code signing certificate:

1. Go to **(Menu)** > **SIGN+**.
2. Under the **CERTIFICATE ACTIONS**, select **Generate CSR > Code Signing Certificate**.
   The **Generate CSR : Code Signing** page is displayed.

3. In the **Group Details** section, choose the group of certificates you wish to assign the CSR to from the **Assign Group** dropdown.
4. Enter/select the **CSR details**.

Field descriptions for the CSR details section

Fields	Description
***CSR Selection**	Select the key generation of CSR as required. The possible selections are: • AppViewX • HSM.
***Device Type**	Select the type of device as required: Options are: • **HSM Devices** • **ADC Devices**
***Device**	Select the device from the dropdown list.
***Key Handler Name**	Enter the name of the key handler.
***Key Reference Name**	Enter the name of the key reference.
***Common Name**	Name that is to be present in the certificate. **Note:** No special characters allowed except en dash (_) and hyphen (-).
Subject Alternative Name	Enter the alternative subject name. For example, DNS or IP address. **Note:** • Multiple values must be separated by a comma. • The cumulative count SANs appears in the certificate property window from the holistic view.

Fields	Description
***Organization**	The Organization name that to be present in the certificate. This field will be auto-filled and editable based on the configuration in the selected group's policy.
Organization Unit	The Organization Unit name that to be present in the certificate. This field will be auto-filled and editable based on the configuration in the selected group's policy.
Locality	The Locality name that to be present in the certificate. This field will be auto-filled and editable based on the configuration in the selected group's policy.
State	The State name that to be present in the certificate. This field will be auto-filled and editable based on the configuration in the selected group's policy.
***Country**	The Country name that is to be present in the certificate. This field will be auto-filled and editable based on configuration. It must be a two-letter country code (for example, US, and so on).
Email Address	The email contact details of the person responsible for maintaining the certificate. Enter the valid e-mail address.
***Validity**	Enter the number in this field and select the entered validity list to be in **Days**, **Months**, and **Years** from the dropdown lists.
Challenge Password	The challenge password for the certificate. Enter if it is applicable. Password must contain at least one alphabet (uppercase and lowercase), one number, and one special character.

Fields	Description
Confirm Password	The password to confirm the Challenge Password entered and match with the Challenge Password.
***Hash Function**	The Hash function with which the CSR has to be signed. For Microsoft Enterprise CA, the targeted CA decides the hash function while issuing the certificate. This field will be auto-filled and editable based on the configuration in the selected group's policy.
***Key Type**	The key type is used while creating a private and public key pair. This field will be auto-filled and editable based on the configuration in the selected group's policy.
***Bit Length**	The bit length is used while creating a private and public key pair. This field will be auto-filled and editable based on the configuration in the selected group's policy.
*: *Mandatory fields*	

5. In the **Attachments** section, upload any attachments relevant to the CSR generation process.

Field descriptions for the Attachments section

Field	Description
Name	Enter the alternate name for the document to be uploaded.
Comments	Enter the comments in this field.  **Note:** You can enter a maximum of 2000 words in the field.
Upload File	Click the **Upload** button to select the file.

Field	Description
	Note: Maintains if there are any additional documents to be maintained in AppViewX. These documents will not be submitted to CA. It is a non-mandatory section.

Tip: You can use the **Search** option to find the attachments from the attachment list.

6. To generate the CSR and add it to the intended group, click **Add**.

Certificate Inventory

The Certificate Inventory allows you to take inventory of, and proactively manage all your certificates. This will be a single source of truth for all the certificates in the organization. Every certificate action can be performed from the inventory.

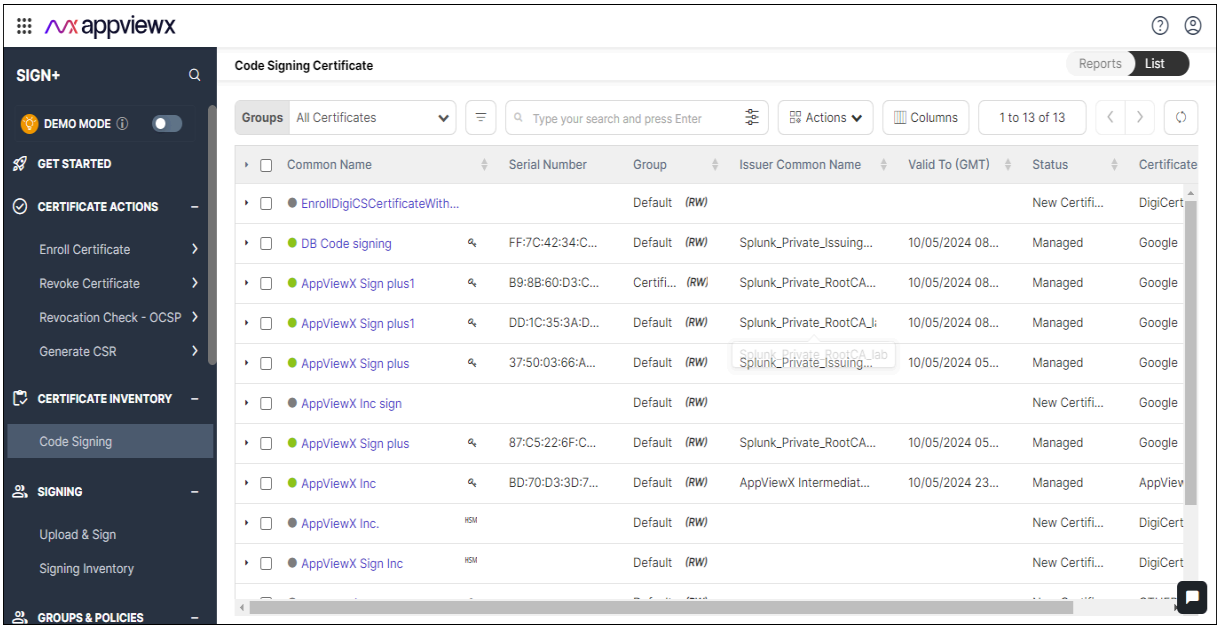
- Code Signing

Code Signing

Accessing Code Signing in **SIGN+**:

1. Go to **(Menu)** > **SIGN+**.
2. Under the **CERTIFICATE INVENTORY**, select **Code Signing**.

The **Code Signing Certificate** page is displayed.



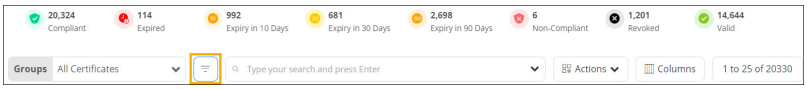
- Code Signing Certificate Inventory
- Downloading Certificate
- Downloading Key
- Downloading CSR
- Certificate Regeneration
- Certificate Renewal
- Certificate Revoke
- Suspend Certificate
- Deleting Code Signing Certificate

Code Signing Certificate Inventory

Code signing certificate inventory displays all the code certificates with the EKU (Extended/Enhanced Key Usage) code signing present. The certificates in this inventory will be shown to the user only based on role-based access control on the certificate group. From this inventory, the user can select one or many certificates and perform bulk certificate revocation checks, search and filter certificates, export certificates, download certificates, delete certificates, and so on.

The **Code Signing Certificate** page is available under **Certificate Inventory** in the left menu.

Options available on the Code Signing Certificate page

Options	Description
Groups	Expanding this dropdown displays the certificate groups and the number of certificates in each group. Selecting a group will display the filtered list of certificates.
Filter Summary	Displays the status of certificates according to expiry, compliance, validity, and so on. 
Advanced Search	Allows you to perform a quick search for specific data. Clicking on the search bar dropdown opens the **Advanced Search** window.
Actions	Displays the list of actions you can perform on the certificates.
Columns	Allows you to select the columns to be displayed on the code signing certificate inventory page.
Toggle	Allows you to toggle between the following display options for the code signing certificate inventory: • **List**: Displays the list of code signing certificates.

- Exporting Code Signing Certificates
- Downloading Code Signing Certificates
- Deleting Code Signing Certificates
- Changing Code Signing Certificate Status
- Assigning Code Signing Certificate Group
- Unassigning Code Signing Certificate Group
- Add/Modify Comments for Code Signing Certificate
- Updating Certificate Attributes for Code Signing Certificate
- Revocation Check for Code Signing Certificates

Exporting Code Signing Certificates

Export certificate action allows the user to export certificate details in the form of columns and values. The user can export all the certificates in the inventory or select only specific certificates and export. The

output of this action can be selected in **.xls** or **.csv** format. This can be used for reporting or creating an inventory.

1. On the **Code Signing Certificate** page, select the certificate that you want to export.
2. From the **Actions** dropdown menu, select **Export Certificates**.

   The **Export** dialog box is displayed.
3. Select the required **Options** and **Format**.

   The selected certificate is exported to your local machine.

Downloading Code Signing Certificates

Code Signing certificates can be downloaded via holistic view only one certificate at a time in multiple formats as PEM, DER, PKCS#7, PKCS#12, and JKS. PKCS#12 and JKS can be downloaded only with the password-protected certificate.

To download code signing certificate:

1. On the **Code Signing Certificate** page, select the certificate(s) that you want to download.
2. From the **Actions**dropdown, click **Download Certificates**.

   The **Download Certificates** pop-up window is displayed.
   a. In the **Download Certificate** pop-up window, select **Certificates Only** or **Certificates and Keys**.
   b. You can also enable/disable **Download Truststore Certificates** option along with the end certificates.

   > **Note:** If you have permission to view the restricted content mentioned in Step, the certificate details are downloaded with <.zip> file. If you do not have the necessary permissions, the system creates and downloads an empty <.zip> file to the destination you specify.

   c. The system enables the **Secret Passphrase** field when you select the **Certificates and Keys** option. Enter a passphrase to encrypt the contents into a *<.ZIP>* file.
3. In the **Download Certificate** dialog box, enter or select the requested field information.

Field Description for Download Certificate

Field	Description
Choose Download Type	Select the certificate download type as: • **Certificate Only** • **Certificate and Keys**
Download Truststore Certificates	Turn on this toggle to download truststore certificates.
Set Password for Keystore	Enter a passphrase to encrypt the contents into a **.zip** file.

a. In the **Download Certificate** pop-up window, select **Certificates Only** or **Certificates and Keys**.
b. You can also enable/disable **the Download Truststore Certificates** option along with the end certificates.

> **Note:** If you have permission to view the restricted content mentioned in Step, the certificate details are downloaded with <.zip> file. If you do not have the necessary permissions, the system creates and downloads an empty <.zip> file to the destination you specify.

c. The system enables the **Secret Passphrase** field when you select **Certificates and Keys**. Enter a passphrase to encrypt the contents into a <.zip> file.

4. Click **Download**.
5. To view details of the certificate, unzip the file and open the security certificate file.
6. Click **Details**.

Deleting Code Signing Certificates

1. On the **Code Signing Certificate** page, select the certificate that you want to delete.
2. From the **Actions** dropdown menu, select **Delete**.

   The **Delete Certificate** dialog box is displayed.

3. Click **Yes**.

   The selected certificate(s) will be deleted.

Changing Code Signing Certificate Status

The status of a certificate can be set as monitored or managed during or after the certificate discovery process and also from the certificate inventory directly. When the certificates are set as Monitored, you can only view the certificate details in reports and in the inventory. When the certificates are set as Managed, the certificate-related actions, along with push/bind operations, can be performed, along with viewing of the certificates in the reports and inventory.

To change the code signing certificate status:

1. On the **Code Signing Certificate** page, select the certificate for which you want to change status.
2. From the **Actions** dropdown, click **Change Status**.
3. In the **Change Status** dialog box that is displayed, in the **Change Status to** field, select the status of the certificate field as **Managed** or **Monitored**.
4. Enter the reason for changing the status, if required, and click **Yes**.

   The certificate status is changed as per the selection.

Assigning Code Signing Certificate Group

The certificates with common attributes can be grouped together to perform compliance checks against policy details, to enable auto-renewal and auto-push operations. You can also view certificates as groups.

To assign a code signing certificate to a group:

1. On the **Code Signing Certificate** page, select the certificate that you want to assign to a group.
2. From the **Actions** dropdown, click **Assign Group**.
3. In the **Assign to Group** dialog box that is displayed, search for the group that you want to assign the certificate.
4. Select the required certificate group.
5. Enter a reason for assigning the certificate to the selected certificate group, if required, and click **Assign**.
   The certificate is assigned to the selected group.

Unassigning Code Signing Certificate Group

You can unassign any certificates from a specific certificate group to the default group. The policy and actions of the default group will be applied to these certificates.

To unassign a code signing certificate from a certificate group:

1. On the **Code Signing Certificate** page, select the certificate that you want to unassign from a certificate group.
2. From the **Actions** dropdown, click **Unassign Group**.
3. In the **Unassign Group** window that is displayed, enter the reason for unassigning the certificate from the group and click **Unassign**.
   The selected certificate is now assigned to the default group.

Add/Modify Comments for Code Signing Certificate

To add/modify comments for certificate(s):

1. On the **Code Signing Certificate** page, select the certificate that you want to revoke.
2. From the **Actions** dropdown, click **Add/Modify Comments**.
3. In the **Add/Modify Comments** pop-up window that is displayed, enter a comment and click **Save**.

Updating Certificate Attributes for Code Signing Certificate

Other than the fields that are defined for CSR, you can add organization-specific values to a request. These values will not be part of the certificate but will be available in the AppViewX inventory. For example, a cost center Inventory can be filtered based on these attributes.

To update the certificate attribute for a code signing certificate:

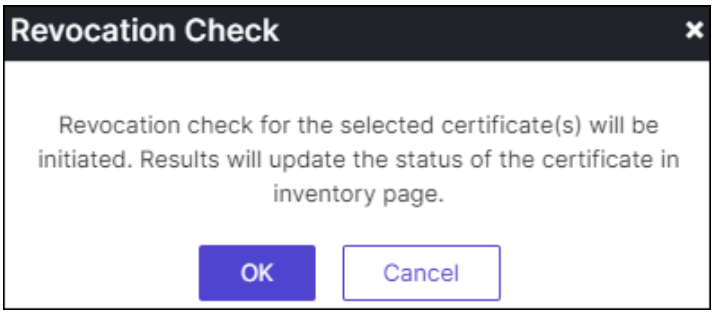
1. On the **Code Signing Certificate** page, select the certificate for which you want to update attributes.
2. From the **Actions** dropdown, click **Certificate Attributes**.
3. Update the **Certificate Attributes** and click **Save**.

Revocation Check for Code Signing Certificates

To perform revocation check for a code signing certificate:

1. On the **Code Signing Certificate** page, select the certificate for which you want to perform a revocation check under **Common Name**.
2. Click **Actions** menu and choose **Revocation Check** from the dropdown.

   In the Revocation check dialog box is displayed.



3. Click **OK** to proceed revocation check.

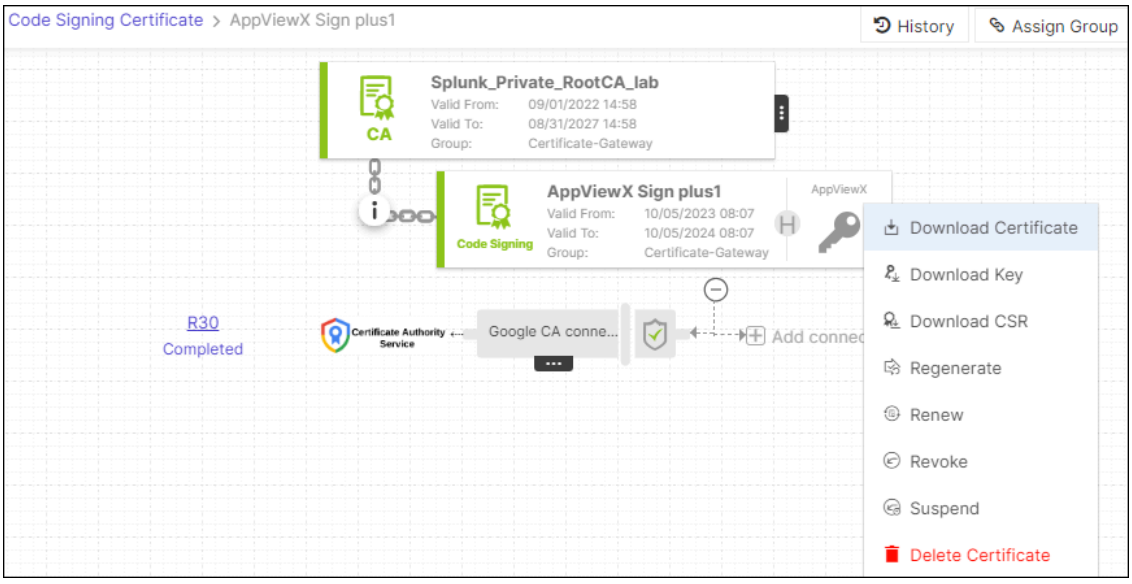
   Revocation Check results will be updated on the inventory page.

Downloading Certificate

Code Signing certificates can be downloaded via holistic view only one certificate at a time in multiple formats as PEM, DER, PKCS#7, PKCS#12, and JKS. PKCS#12 and JKS can be downloaded only with the password-protected certificate.

To download code signing certificate:

1. On the **Code Signing Certificate** page, select the certificate that you want to download.
2. Hover over the three-dot menu for the certificate, and then click **Download Certificate**.



3. The **Download Certificates** pop-up window is displayed.

a. In the **Download Certificate** pop-up window, select **Certificates Type**.
b. You can also enable/disable **Download Truststore Certificates** option along with the end certificates.

> **Note:** If you have permission to view the restricted content mentioned in Step, the certificate details are downloaded with <.zip> file. If you do not have the necessary permissions, the system creates and downloads an empty <.zip> file to the destination you specify.

c. The system enables the **Secret Passphrase** field when you select the **Certificates and Keys** option. Enter a passphrase to encrypt the contents into a *<.ZIP>* file.

4. In the **Download Certificate** dialog box, enter or select the requested field information.

Field Description for Download Certificate

Field	Description
Choose Download Type	Select the certificate download type as: • **Certificate Only** • **Certificate and Keys**
Download Truststore Certificates	Turn on this toggle to download truststore certificates.
Set Password for Keystore	Enter a passphrase to encrypt the contents into a **.zip** file.

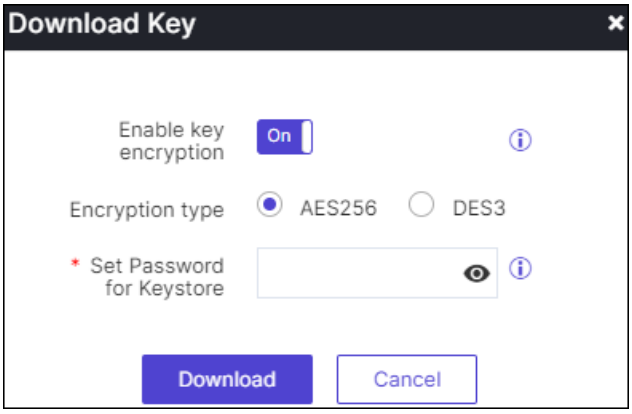
5. Click **Yes**.
6. To view details of the certificate, unzip the file and open the security certificate file.
7. Click **Details**.

Downloading Key

To download key for code signing certificate:

1. On the **Code Signing Certificate** page, select the certificate for which you want to download the key.
2. Hover over the three-dot menu for the certificate, and then click **Download Key**.

3. The **Download Key** pop-up window is displayed.



4. If the **Enable key encryption** is enabled, Encryption type option is displayed select the required encryption type.

   When the **Enable key encryption** is enabled, the **Encryption type** option becomes visible for selecting the desired encryption type.

5. **Set Password for Keystore**to encrypt the content into .ZIP file.
6. Click **Download**.

   The selected certificate key will be Downloaded.

Downloading CSR

1. On the **Code Signing Certificate** page, select the certificate for which you want to download CSR.
2. Hover over the three-dot menu for the certificate, and then click **Download CSR**.

The certificate CSR will be downloaded.

Certificate Regeneration

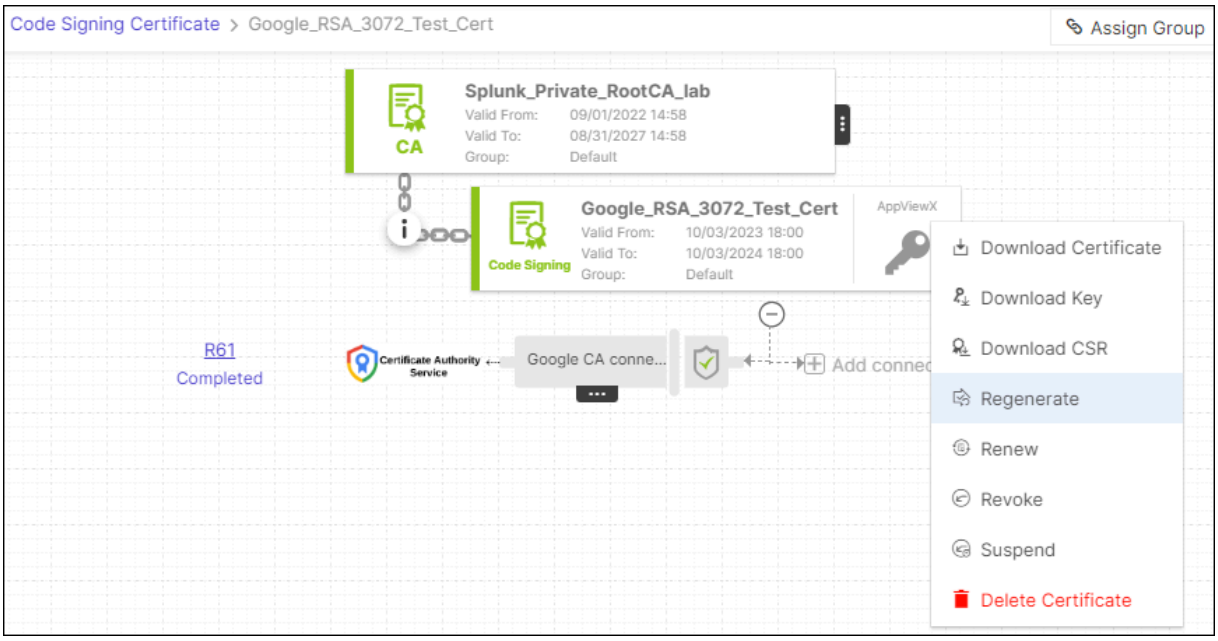
Certificate regeneration involves the process of generating a new certificate that replicates the parameters of an existing certificate. Regenerating a certificate entails placing a new order with the Certificate Authority (CA). This option becomes particularly useful when a user intends to transition to a different CA for certificate issuance. During this process, a new Certificate Signing Request (CSR) will be submitted to the CA.

To regenerate Code Signing Certificate, follow these steps:

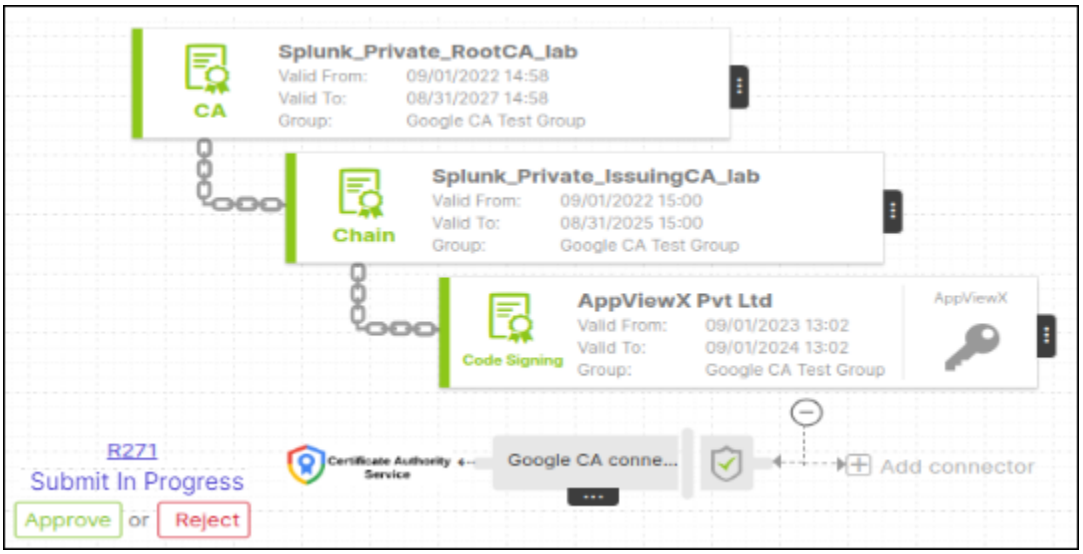
1. Go to **(Menu)** > **SIGN+**.
2. Under the **CERTIFICATE INVENTORY**, select **Code Signing**.

   The **Code Signing Certificate** page is displayed.
3. To renew a **Code Signing Certificate**, select the certificate name under **Common Name**.

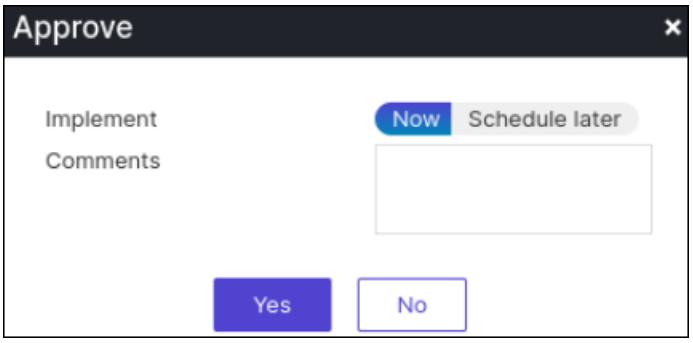
   The holistic view of the selected certificate is displayed.
4. Hover over the three-dot menu for the certificate, and then click **Regenerate**.



5. You will be redirected to the **Code Signing Certificate** > **Regenerate** page.
6. Proceed with the necessary modifications of the required details in the **General Information, CA Details, CSR Parameters, Attachments, Generic Fields** section.
7. Click **Regenerate**.
8. The regenerate process is initiated. On the certificate's holistic view, click **Approve**to proceed.

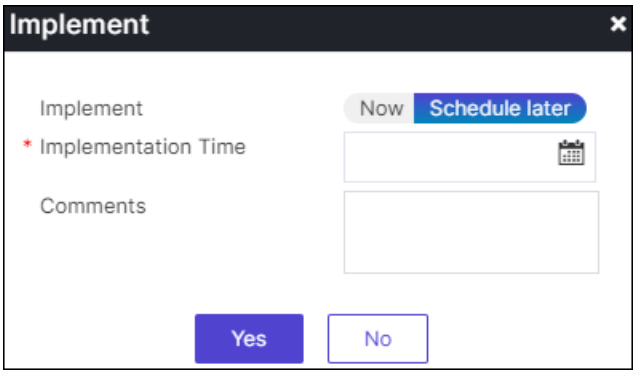


9. In the **Approve** dialog box, enter your comments (optional).



If you want to schedule automatic approval for the workflow request in the future, click **Schedule later**. You can then choose the Implementation Time from the calendar field.

10. Click **Yes**.
11. On the certificate holistic view, click **Implement** to proceed.
12. In the **Implement** dialog box, enter your comments.



If you want the workflow request to be automatically approved in the future, click **Schedule later**. You can then select the **Implementation Time** from the calendar field.

13. Click **Yes**.
14. After the certificate is regenerated, the status updates to **Completed**.

Certificate Renewal

Code Signing Certificates are issued with a limited validity period. Before the expiration of their validity, the certificates have to be renewed for service continuity. The renewal process is specific to CAs, depending on their operations. The result is the issuance of a certificate with extended validity. SIGN+ enables you to trigger certificate renewals. You can trigger renewal from the certificate's holistic view. During this process, an old Certificate Signing Request (CSR) will be submitted to the CA.

To renew Code Signing Certificate, follow these steps:

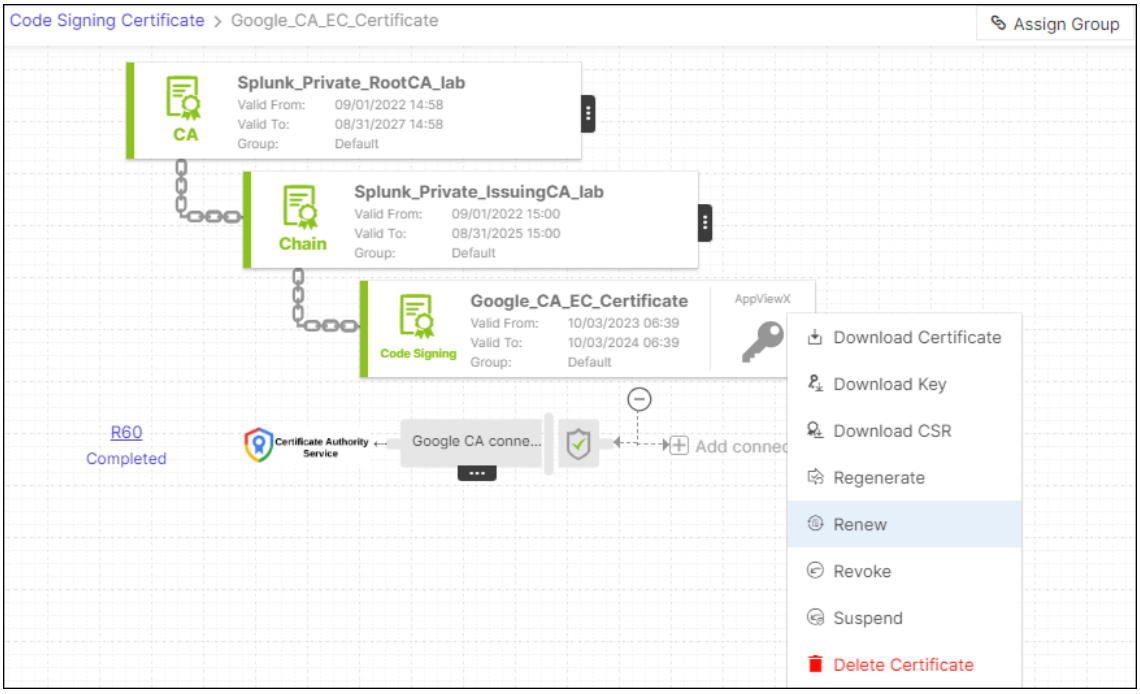
1. Go to **(Menu)** > **SIGN+**.
2. Under the **CERTIFICATE INVENTORY**, select **Code Signing**.

   The **Code Signing Certificate** page is displayed.

3. To renew a **Code Signing Certificate**, select the certificate name under **Common Name**.

   The holistic view of the selected certificate is displayed.

4. Hover over the three-dot menu for the certificate, and then click **Renew**.



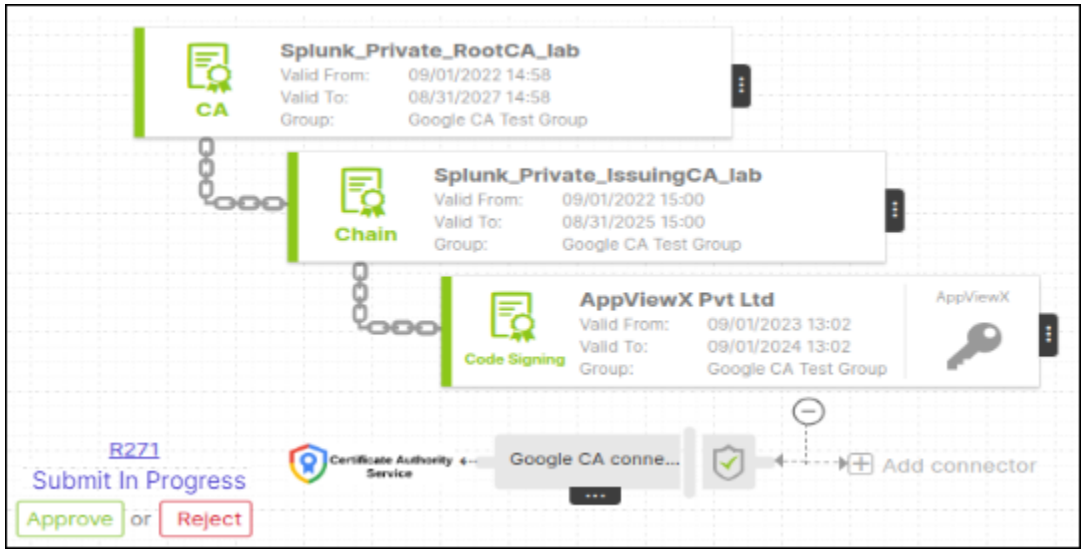
5. You will be redirected to the **Code Signing Certificate** > **Renew** page.
6. Proceed with the necessary modifications of the required details in the **General Information, CA Details, CSR Parameters, Attachments, Generic Fields** section.
7. Click **Renew**.

   The **Renew** dialog box is displayed.

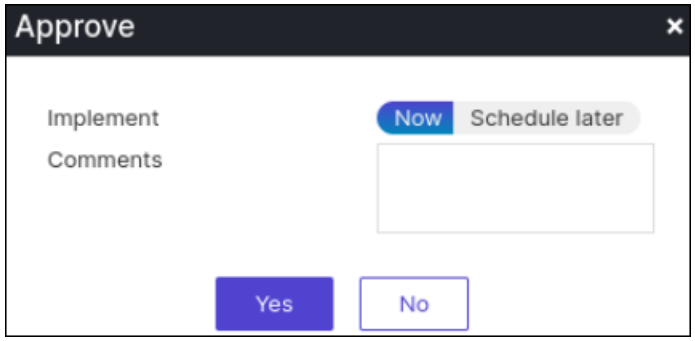
8. Enter your comments in the text field and click **Yes**.

   This action automatically generates a request ID, also known as the work order ID. The work order status is displayed next to the certificate in the holistic view. If the 'approval required' option is enabled in the CA policy, the request will progress to the **Approve** and **Implementation** stages.

9. The renewal process is initiated. On the certificate's holistic view, click **Approve** to proceed.

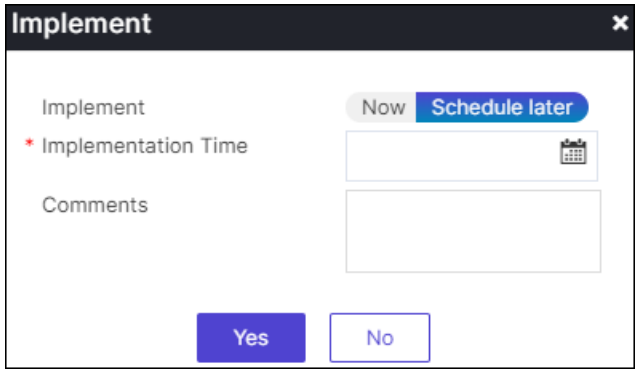


10. In the **Approve** dialog box, enter your comments (optional).



If you want to schedule automatic approval for the workflow request in the future, click **Schedule later**. You can then choose the Implementation Time from the calendar field.

11. Click **Yes**.
12. On the certificate holistic view, click **Implement** to proceed.
13. In the **Implement** dialog box, enter your comments.



If you want the workflow request to be automatically approved in the future, click **Schedule later**. You can then select the **Implementation Time** from the calendar field.

14. Click **Yes**.
15. The renewal process is initiated. Once the renewal is finished, the workflow status will be updated to **Completed**.

Certificate Revoke

Revocation is the process of making a certificate invalid. For example, you might need to revoke a certificate if the certificate is no longer required or the certificate's private key is compromised. Make sure that you have permission to revoke a certificate and submit a request to the certificate authority. As soon as the certificate is revoked, it is not considered to be a trusted certificate. Revoked certificates are listed in the Certificate Revocation List (CRL) maintained by each certificate authority.

To revoke Code Signing Certificate, follow these steps:

1. Go to **(Menu)** > **SIGN+**.
2. Under the **CERTIFICATE INVENTORY**, select **Code Signing**.

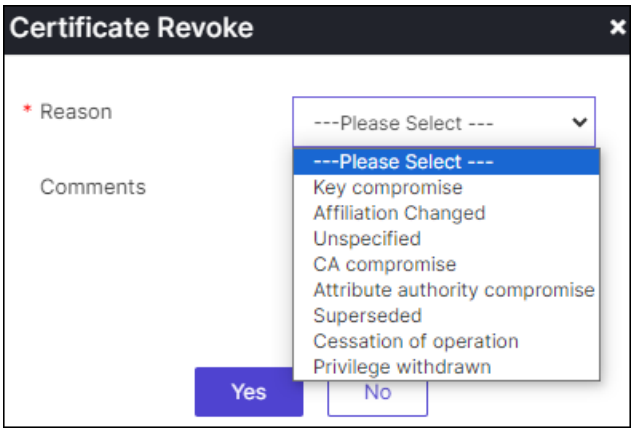
   The **Code Signing Certificate** page is displayed.

3. To revoke a **Code Signing Certificate**, select the certificate name under **Common Name**.

   The holistic view of the selected certificate is displayed.

4. Hover over the three-dot menu for the certificate, and then click **Revoke**.

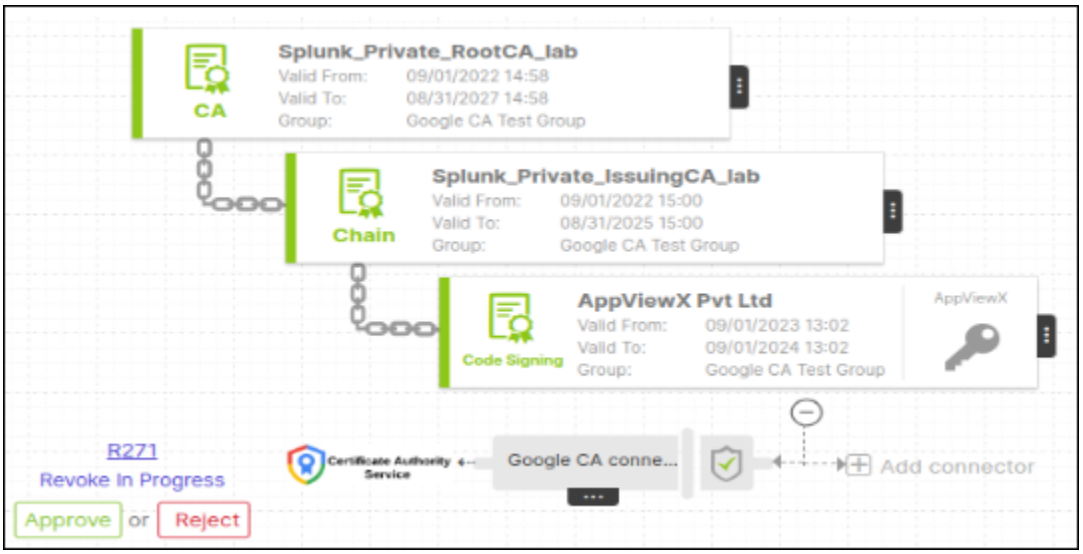
5. The **Certificate Revoke** dialog box is displayed.
6. In the **Certificate Revoke** dialog box, select the reason for revoking the certificate from the dropdown list.
7. You can also enter comments in the Comments text field (optional), and then click **Yes**.



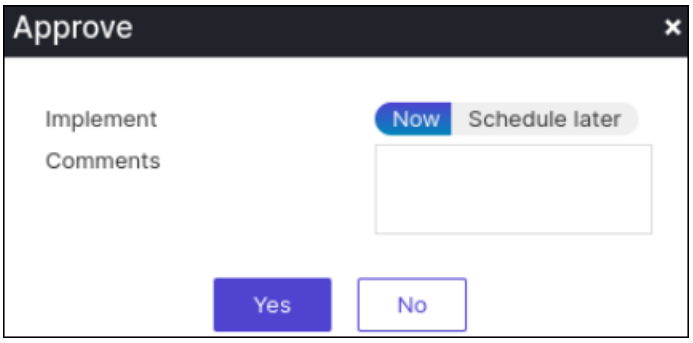
8. The revoke process is initiated for the selected certificate.

   If the **Approval Required** checkbox is enabled, the request will progress through the **Approve** and **Implementation** stages.

9. To proceed, click **Approve** on the certificate's holistic view.

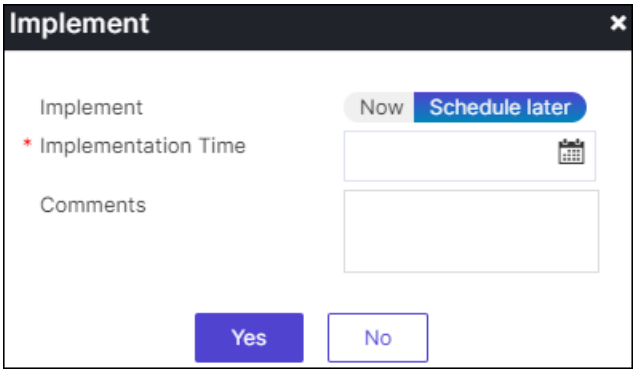


10. In the **Approve** dialog box, enter your comments (optional).



If you want to schedule automatic approval for the workflow request in the future, click **Schedule later**. You can then choose the Implementation Time from the calendar field.

11. Click **Yes**.
12. On the certificate holistic view, click **Implement** to proceed.
13. In the **Implement** dialog box, enter your comments.



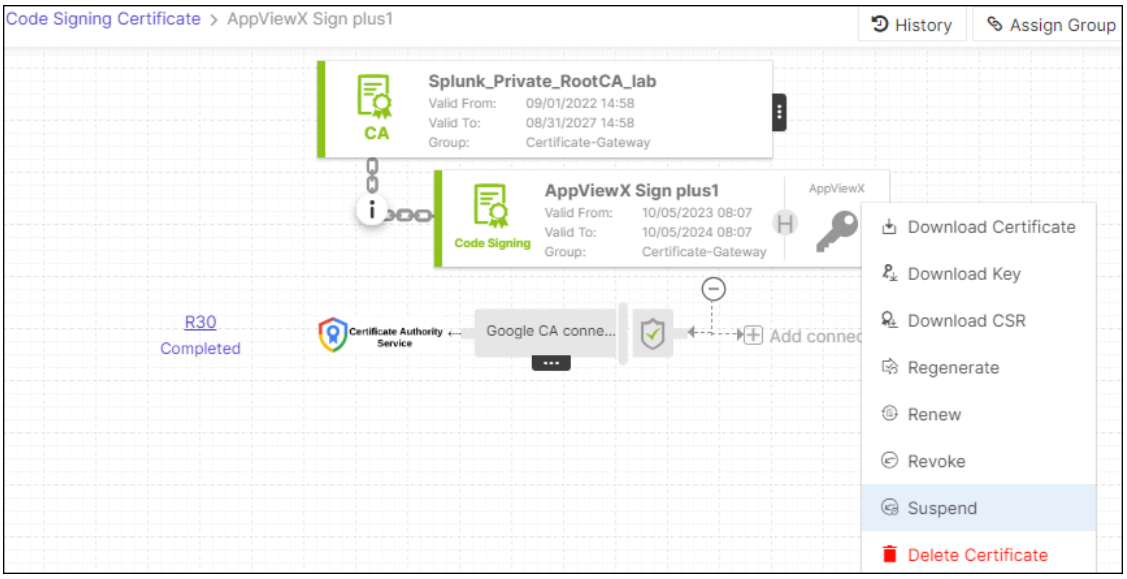
If you want the workflow request to be automatically approved in the future, click **Schedule later**. You can then select the **Implementation Time** from the calendar field.

14. Click **Yes**.
15. After the certificate is revoked, the status updates to **Completed**.

Suspend Certificate

Follow the steps below to suspend certificate:

1. On the **Code Signing Certificate** page, select the certificate that you want to suspend.
2. Hover over the three-dot menu for the certificate, and then click **Suspend**.

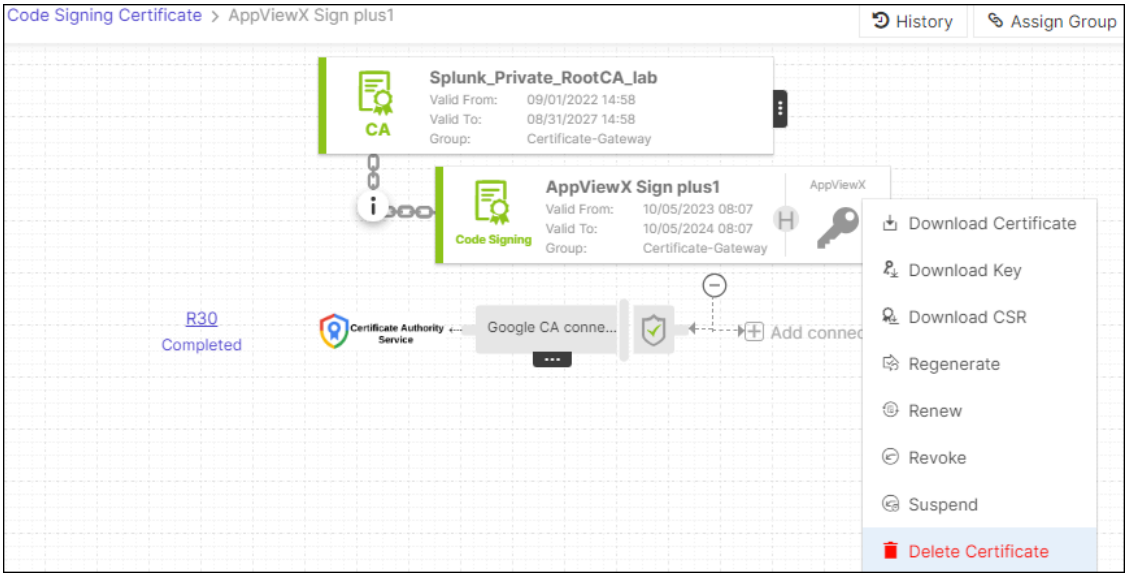


3. In the **Certificate Suspend** dialog box, select the reason for suspending the certificate from the dropdown.
4. You can also enter comments in the comments text field (optional), and then click **Yes**.

   The selected certificate will be suspended.

Deleting Code Signing Certificate

1. On the **Code Signing Certificate** page, select the certificate that you want to delete.
2. Hover over the three-dot menu for the certificate, and then click **Delete Certificate**.



3. Delete Certficate **Confirmation** dialog box is displayed.
4. Click **Yes**.

The selected certificate will be deleted.

Signing Inventory

The Signing Inventory functions as a centralized repository that consolidates all your signed files. It offers a comprehensive overview of each signed asset, encompassing crucial details such as the signing timestamp, the applied signing policy, the certificate key used for signing, timestamping specifics, file type, signing type (HASH-based or FILE-based), associated username, and the IP address used during signing. Moreover, it provides visibility into the status of each file, indicating whether it has been successfully signed or not.

This feature empowers you to efficiently manage and monitor your signed files, with the ability to download or delete them as needed, making it a valuable tool for your signing process management.

You can modify the visibility of signing requests, displaying either all requests to all users or only signed requests to the logged-in user.

To make these adjustments, modify the ACF permissions within **Platform** > **Role** settings. Under **Signing Inventory**:

1. Select **View All Signing** to see all signed requests.
2. Select **View My Signing** to view only signed requests by the logged-in user.

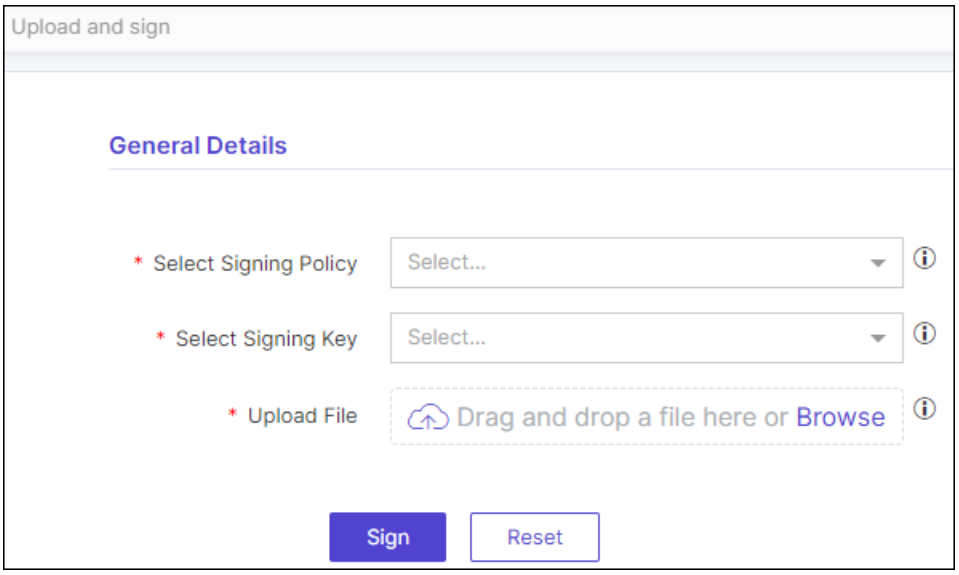
Upload and Sign

The **Upload and Sign** page enables users to upload a file for digital signing. On this page, users can choose from a selection of signing policies and the associated certificate keys to sign their uploaded files. Some signing policies may include specific meta-information configurations, ensuring that the signed files adhere to predefined policy requirements.

To **Upload and Sign** a file using Code Signing Certificate, follow these steps:

1. Go to **(Menu) > SIGN+ > SIGNING > Upload & Sign**.

   The **Upload and Sign** page is displayed.



2. Enter the required details, under **General Details** section.
   a. **Select Signing Policy:** Select the suitable signing policy according to your specific needs, considering any configurations applied during the signing policy's creation.

   Note: The default policy name is set as the first policy in the dropdown.

   b. **Select Signing Key:** Select the signing key that corresponds to the policy selected in the previous step.

   Note: The default signing key is set as the signing key of first policy in the dropdown.

   c. For **specific policies**, you may need to provide **additional information**. Kindly enter these details as configured in the selected signing policy.
   d. **Upload File**: Only selected file types during policy creation are allowed for upload.

   Note: The JSign version v6.0 is currently in use for signing to support a wider range of file types.

3. Click **Sign** to initiate the signing process.
4. After the signing process is complete, the file will appear in the Signing Inventory, where you can proceed to download it.

What to do next:

- Download Code Signed file that have been digitally signed and verified.

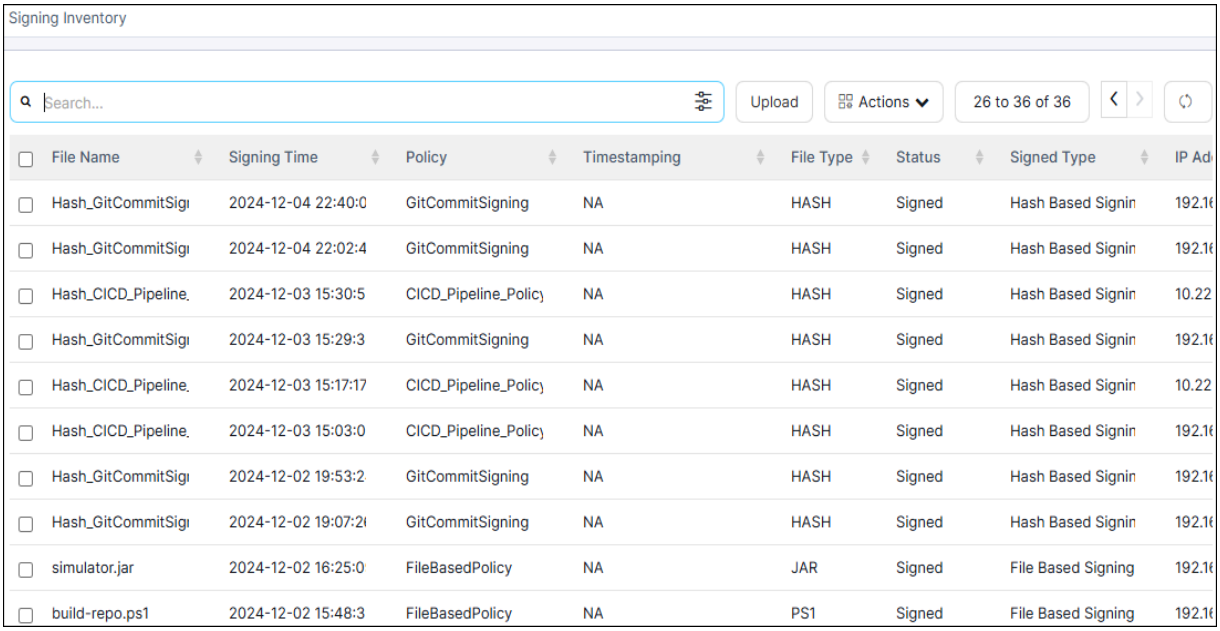
Accessing Signing Inventory

You have to access the **SIGN+** node to access the various functions provided by it.

To access Signing Inventory:

1. Go to **(Menu) > SIGN+ > Signing > Signing Inventory**.

   The **Signing Inventory** page is displayed.



2. Select one or more inventory files to be **Downloaded** or **Deleted**.
3. From the command bar on the top right, click **Actions**.
4. From the list of available actions, select **Download** or **Delete**.
5. To **Download** or **Delete** the selected file, click **Yes** in the **Confirm download** or **delete** pop-up window.
6. The selected file will be **Downloaded** or **Deleted**.

7. Additionally, you have the option to **Upload** file.
8. From the command bar, click **Upload**.

   The **Upload & Sign** page is dispalyed, where you can proceed with file upload.

9. To select a file to be uploaded to the signing inventory, click **Browse**.
10. Select the required file and click **Upload**.

• Advanced Search

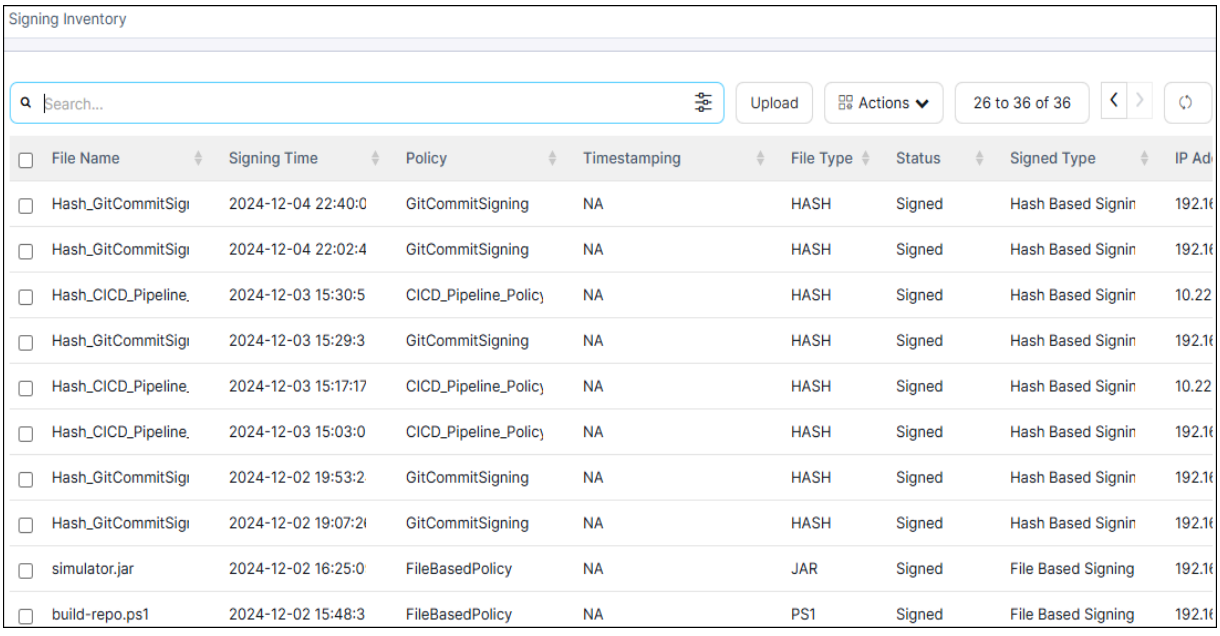
• Exporting Signed Entries from Signing Inventory

Advanced Search

The Advanced Search enables users to quickly search for specific data. That allows users to apply filters, narrowing down the view within the signing inventory to display only the most relevant data. It helps users navigate large datasets more efficiently.

1. Go to **(Menu) > SIGN+ > Signing > Signing Inventory**.

   The **Signing Inventory** page is displayed.



2. Click **(Filter)** icon on the search bar.
   The **Advanced Search** window appears.

3. Enter the necessary details based on your search criteria, such as File Name, File Type, Status, Signing Time Period, Policy, Signing Key, and Signed Type.
4. Click **Search**.
   The requested search results are displayed in the Signing Inventory.

Whats Next ?

- Export the signed entries from the signing inventory.

Exporting Signed Entries from Signing Inventory

Exporting signed entries from signing inventory allows you to export selected signing data, usually for reporting, analysis, or archival purposes. This simplifies managing and sharing code signed records.

To export signed entries from signing inventory:

1. Go to **(Menu) > SIGN+ > Signing > Signing Inventory**.

   The **Signing Inventory** page is displayed.

   Note: You can also filter the required signed entries from the signing inventory using the advanced search option, and then export them.

2. From the **Actions** dropdown menu, select **Export**.
   The **Export** pop-up window is displayed.
3. Select the required format, columns, and select period for export.
4. Click **Export**.
   The signed entries from the signing inventory will be exported in the selected format.

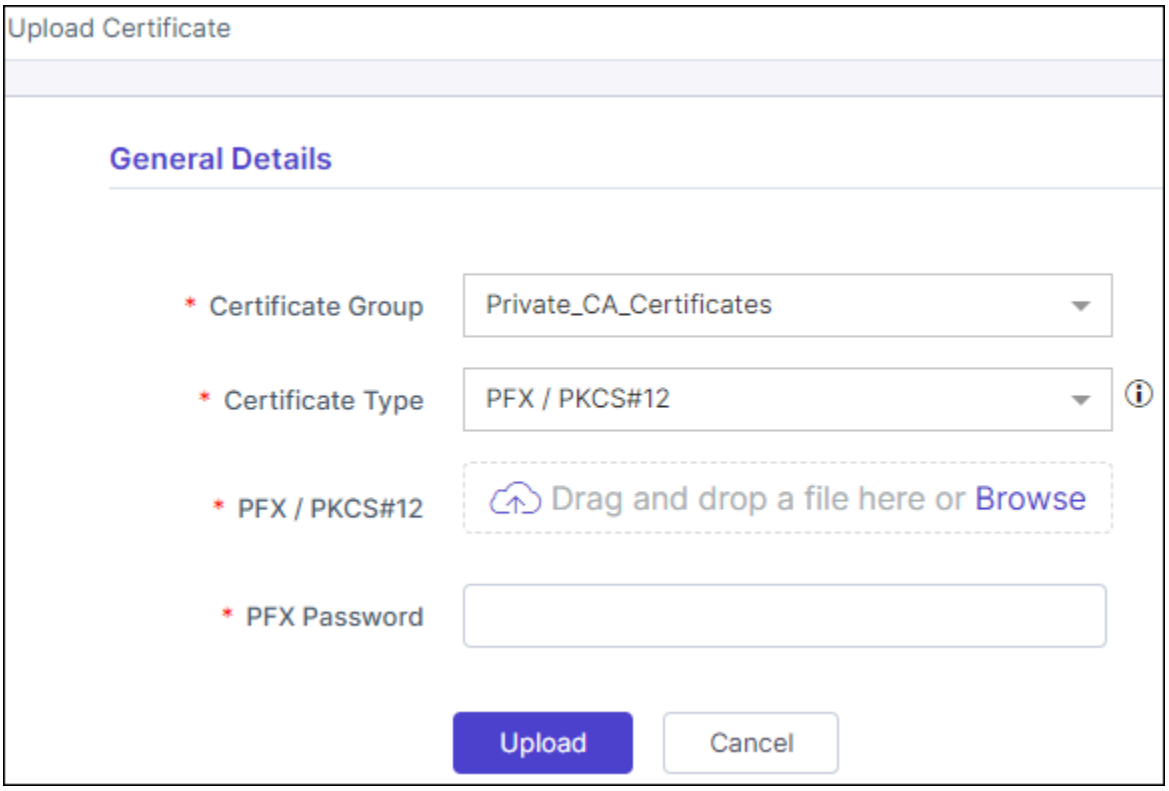
Upload Certificate

The "Upload Certificate" feature enables customers with existing certificates to securely upload them onto the platform. Once uploaded, users can readily configure policy settings and initiate the code signing process using the uploaded certificate. This streamlined process ensures a smooth transition for customers prepared to utilize their existing certificates for code signing.

To upload an existing certificate for code signing, follow these steps:

1. Go to **(Menu)** > **SIGN+** > **SIGNING** > **Upload Certificate**.

   The Upload Certificate page is displayed.



2. Enter the required details, under the **General Details** section.

Field description for General Details

Fields	Description
*** Certificate Group**	Select the appropriate certificate group from the dropdown menu.
*** Certificate Type**	Select the certificate type from the dropdown menu. • **PFX / PKCS#12** PFX / PKCS#12 allows you to upload a codesigning certificate packaged in a PFX or PKCS#12 file, which includes both the certificate and the private key, and is commonly used for importing and exporting certificates and keys. • **Certificate and Key**

Fields	Description
	Certificate and Key is for when you have separate files for the codesigning certificate and the private key, requiring you to upload both files individually. • **Access from HSM.** Access from HSM is used when your private key is stored in a Hardware Security Module (HSM), enhancing security by ensuring the private key never leaves the HSM.
*** PFX / PKCS#12**	**Note:** This field is displayed only when the **Certificate Type** is selected as **PFX / PKCS#12**. Click **Browse** to select the certificate file PFX / PKCS#12 from your local system.
*** PFX Password**	**Note:** This field is displayed only when the **Certificate Type** is selected as **PFX / PKCS#12**. Enter the password associated with the uploaded PFX or PKCS#12 file.
*** Certificate**	**Note:** This field is displayed only when the **Certificate Type** is selected as **Certificate and Key**. Click **Browse** to select the certificate file from your local system.
*** Key**	**Note:** This field is displayed only when the **Certificate Type** is selected as **Certificate and Key**. Click **Browse** to select the key file from your local system. If the key is password-protected, users may be prompted to enter the correct password for successful key upload alongside the certificate. Failure to provide the correct password may result in key upload failure, while the certificate upload will proceed successfully.

Fields	Description
*** HSM Vendor**	**Note:** This field is displayed only when the **Certificate Type** is selected as **Access from HSM**. Select the vendor of your HSM from the dropdown menu.
*** Certificate Source**	**Note:** This field is displayed only when the **Certificate Type** is selected as **Access from HSM**. Select the appropriate **Certificate Source**. • **Upload:** Click browse and upload the certificate from your local system. • **Pick from HSM:** Choose this option to select a certificate from the dropdown menu. **Note:** After selecting this option, you can click **(Refresh)** icon to update the list of available certificates.
*** Certificate**	**Note:** This field is displayed only when the **Certificate Type** is selected as **Access from HSM**. • If you select **Upload** as the Certificate Source, then click **Browse** to select the certificate from your local system. • If you select **Pick from HSM** as the Certificate Source, then select the certificate from the dropdown menu.
Private Key Label	**Note:** This field is displayed only when the **Certificate Source** is selected as **Upload**. Enter the private key label associated with the uploaded certificate. If you are aware of the precise private key label for the certificate you intend to upload, please provide it. Otherwise, failure may occur during the Code Signing Process.

Fields	Description
*: *Mandatory fields*	

3. Click **Upload** to initiate the certificate upload process.
   Upon successful upload, a confirmation message will be displayed, and the certificate will be added to the Signing Inventory.

What to do next:

- Configure the signing policy using the uploaded certificate.
- **Code signing** after configuring the policy.

Integration with CI/CD pipeline

SIGN+ Integration with CI/CD pipeline refers to the seamless inclusion of code signing processes using SIGN+ (a code signing solution) within a continuous integration and continuous deployment pipeline, ensuring that all code releases are securely signed before deployment.

- Need for Code Signing
- Integrating Code Signing in Jenkins Pipeline
- Integrating Code Signing in GitLab Pipeline
- Integrating Code Signing in Azure Devops Pipeline
- Integrating Code Signing in GitHub Actions Pipeline
- Integrating Code Signing in Atlassian Bamboo Pipeline
- Integrating Code Signing in AWS DevOps
- Appendix

Need for Code Signing

CI/CD is a Software Development Life Cycle (SDLC) process that supports agile development methodologies designed to deliver software in frequent release cycles rapidly and with high quality.

When code or software is ready for production, it is released to organizations and departments for installation on their systems. Prior to release, the code can be signed to verify the identity of the publisher and ensure it hasn't been altered. This increases the authenticity and integrity of the code, enhancing trust and security.

Integration with CI/CD tools automates code signing, ensuring compliance with security policies without slowing down development. This improves the overall efficiency of the process. Additionally, signing code from external sources included in the CI/CD process provides assurance and trust in their inclusion within the software development process.

Supported OS Versions

Linux: AppViewX PKCS11 supports CI/CD pipeline integration with build servers hosted on the following Linux OS versions:

1. Ubuntu 20.04.6 LTS
2. Fedora - Red Hat Enterprise Linux 8.7 (Ootpa)
3. SUSE Linux Enterprise Server 15 SP5
4. Amazon Linux
5. Debian

Windows: AppViewX CSP/PKCS11 supports CI/CD pipeline integration with build servers hosted on Windows OS versions.

Note: AppViewX PKCS11 does not support CentOS, as it has reached its end of life.

Integrating Code Signing in Jenkins Pipeline

Jenkins

Jenkins is a self-contained, open-source automation server that can be used to automate all sorts of tasks related to building, testing, and delivering or deploying software.

Jenkins can be installed through native system packages, Docker, or even run standalone by any machine with a Java Runtime Environment (JRE) installed.

Jenkins Pipeline

Jenkins Pipeline is a suite of plugins that support implementing and integrating continuous delivery pipelines into Jenkins. Pipeline adds a powerful set of automation tools to Jenkins, supporting use cases that span from simple continuous integration to comprehensive CD pipelines.

Pipeline provides an extensible set of tools for modeling simple-to-complex delivery pipelines "as code" via the Pipeline domain-specific language (DSL) syntax.

Jenkinsfile

A Jenkinsfile is a text file that defines the entire pipeline of a Jenkins job or build process. It allows you to define the various stages, their order, and the actions or steps to be executed within each stage. A sample Jenkins file content is as follows:

```
pipeline {
    agent any
    stages {
        stage('Build') {
            steps {
                //
            }
        }
        stage('Test') {
            steps {
                //
            }
        }
        stage('Deploy') {
            steps {
                //
            }
        }
    }
}
```

Code Signing Integration with Native Tools using AppViewX SIGN+ in Jenkins Pipeline

Prerequisites

A repository with Jenkins pipeline setup in the runner. Download the SIGN+_Package.zip for the required OS and install in the required build server/runner and ensure connectivity from the build server/runner to the SIGN+ API Connector URL.

Note: The SIGN+_Package should be installed under the same user as which the pipeline job is getting triggered.

Code Signing Integration with AppViewX CSP/PKCS#11

Using Signtool with AppViewX CSP

1. Run the AppViewX SIGN+ Installer executable to install the prerequisites required to use the AppViewX CSP/PKCS11 Providers.
2. Copy the signtool command generated in the README File and update the Jenkins Configuration File with the appropriate script.

```
stage('sign') {
  steps {
   bat 'signtool.exe sign /f <path to certificate>  /fd <digest algorithm>  /csp <csp_name> /k <key_alias_name> /tr <timestamp_url> /td <timestamp digest
 algorithm>  <input_file_path>'
}
```

   - **/f <path to certificate>:** Path to your code-signing certificate.
   - **/fd <digest algorithm>:** Specifies the hashing algorithm.
   - **/csp <csp_name>:** Name of Cryptographic Service Provider (CSP).
   - **/k <key_alias_name>:** Key Container Name
   - **/tr <timestamp_url>:** Provides a timestamp from a trusted timestamping authority.
   - **/tr <timestamp_digest>:** Specifies the timestamping Digest algorithm.
   - **<input_file_path>:** Path to the file to be signed.

   The **<path to certificate>, <digest algorithm>, <csp_name>, <key_alias_name>, <timestamp_url>, <timestamp_digest>** parametersare auto generated based on the signing policy configurations in the README after running the SIGN+ Installer.

Using JarSigner with AppViewX CSP

1. Run the AppViewX SIGN+ Installer executable to install the prerequisites required to use the AppViewX CSP/PKCS11 Providers.
2. Copy the Jarsigner command generated in the README File and update the Jenkins Configuration File with the appropriate script.

```
stage('sign') {
  steps {
    bat 'jarsigner.exe -verbose -storetype "Windows-My" -keyStore NONE -tsa <time_stamp_url> <input_file_path> -signedjar <output_file_path> -sigalg
 <signature algorithm> <keypair alias>'
}
```

   The **<time_stamp_url>, <signature algorithm>** and **<keypair alias>** parameters are auto generated in the README after running the SIGN+ Installer.

Using Nuget with AppViewX CSP

1. Run the AppViewX SIGN+ Installer executable to install the prerequisites required to use the AppViewX CSP/PKCS11 Providers.
2. Copy the nuget command generated in the README File and update the Jenkins Configuration File with the appropriate script.

```
stage('sign') {
  steps {
    bat 'nuget.exe sign <input_file_path> -Timestamper <timestamp_url> -CertificateFingerprint <certificate_fingerprint> -HashAlgorithm <hashing_algorithm>
 -Verbosity detailed -Overwrite'
}
```

The **<time_stamp_url>, <certificate_fingerprint>** and **<hashing_algorithm>** parameters are auto generated in the README after running the SIGN+ Installer.

Using JarSigner with AppViewX PKCS#11 Provider

1. Run the AppViewX SIGN+ Installer executable to install the prerequisites required to use the AppViewX CSP/PKCS11 Providers.
2. Copy the Jarsigner command generated in the README File and update the Jenkins Configuration File with the appropriate script.

```
stage('sign') {
  steps {
    //Windows
    bat 'jarsigner.exe -verbose -keystore NONE -storetype PKCS11 -certs -providerclass sun.security.pkcs11.SunPKCS11 -providerArg <path to
 AVXPKCS11V1.cfg> <input_file_path> -signedjar <output_file_path> -tsa <time_stamp_url> -sigalg <signature algorithm> <keypairalias>'

    //Linux
    bat 'jarsigner -verbose -keystore NONE -storetype PKCS11 -certs -providerclass sun.security.pkcs11.SunPKCS11 -providerArg <path to
 AVXPKCS11V1.cfg> <input_file_path> -signedjar <output_file_path> -tsa <time_stamp_url> -sigalg <signature algorithm> <keypairalias>'

}
```

The **<path to AVXPKCS11V1.cfg>, <time_stamp_url>, <signature algorithm>** and **<keypair alias>** parameters are auto generated in the README after running the SIGN+ Installer.

Using JSign with AppViewX PKCS#11 Provider

1. Run the AppViewX SIGN+ Installer executable to install the prerequisites required to use the AppViewX CSP/PKCS11 Providers.
2. Copy the JSign command generated in the README File and update the Jenkins Configuration File with the script.

```
stage('sign') {
  steps {
    //Windows
    bat 'java -jar <path_to_jsign_jar> --keystore <path to AVXPKCS11V1.cfg> --storetype PKCS11 --storepass 12345678 --alias <keypair alias>  --alg <digest algorithm> --tsaurl <timestamp url> <input_file_path>'

    //Linux
    sh 'java -jar <path_to_jsign_jar> --keystore <path to AVXPKCS11V1.cfg> --storetype PKCS11 --storepass 12345678 --alias <keypair alias>  --alg <digest algorithm> --tsaurl <timestamp url> <input_file_path>'

}
```

The **<path to AVXPKCS11V1.cfg>, <keypair alias>, <digest algorithm>, <timestamp url>** parameters are auto generated based on the signing policy configurations in the README after running the SIGN+ Installer.

Using APKSigner with AppViewX PKCS#11 Provider

1. Run the AppViewX SIGN+ Installer executable to install the prerequisites required to use the AppViewX CSP/PKCS11 Providers.
2. Copy the APKSigner command generated in the README File and update the Github Actions Configuration File with the appropriate script.

```
stage('sign') {
  steps {
    // Windows
    bat 'java -jar <path_to_apk_signer_jar> sign --provider-class sun.security.pkcs11.SunPKCS11 --provider-arg <path to AVXPKCS11V1.cfg> --ks NONE --ks-type PKCS11 --ks-pass pass:12345678 --ks-key-alias <keypair alias> --in "<input_file_path>" --out "<output_file_path>" --v1-signing-enabled false --v2-signing-enabled false --v3-signing-enabled true --v4-signing-enabled false'

    // Linux
```

```
    sh 'java -jar <path_to_apk_signer_jar> sign --provider-class sun.security.pkcs11.SunPKCS11 --provider-arg <path to AVXPKCS11V1.cfg> --ks NONE
  --ks-type PKCS11 --ks-pass pass:12345678 --ks-key-alias <keypair alias> --in "<input_file_path>" --out "<output_file_path>" --v1-signing-enabled false
  --v2-signing-enabled false --v3-signing-enabled true --v4-signing-enabled false'
}
```

The **<path to AVXPKCS11V1.cfg>, <keypair alias>** parameters are auto generated based on the signing policy configurations in the README after running the SIGN+ Installer.

Integrating Code Signing in GitLab Pipeline

GitLab

GitLab is a web-based DevOps platform that provides a complete set of tools for managing the software development lifecycle. It is built on top of the Git version control system and offers features for source code management, continuous integration and deployment (CI/CD), project management, and collaboration.

GitLab Pipeline

In GitLab, a pipeline is a series of stages and jobs that define the steps for building, testing, and deploying your software. It is a core feature of GitLab's CI/CD capabilities.

The pipeline is divided into various stages, and each stage consists of one or more jobs. Agents called GitLab Runners execute the jobs defined in the pipeline when they are triggered by various events such as code pushes, merge requests, etc.

GitLab Configuration File

The GitLab CI/CD configurations are defined in the root repository in a file called **“.gitlab-ci.yml”.**
In the file, you can define the scripts to be run, dependencies, commands to run in order, and other configuration files and templates to be included.

A `.gitlab-ci.yml` file might contain:

```
stages:
  - build
  - test
build-code-job:
  stage: build
  script:
    - echo "Signing Command 1"
```

```
test-code-job1:
 stage: test
 script:
  - echo "Signing Command 2"
```

Code Signing Integration with Native Tools using AppViewX SIGN+ in GitLab Pipeline

Prerequisites

1. A Git repository with GitLab pipeline setup in the runner.
2. Download the **SIGN+_Package.zip** for the required OS and install in the required build server/runner and ensure connectivity from the build server/runner to the SIGN+ API Connector URL.

> **Note:** The SIGN+_Package should be installed under the same user as which the pipeline job is getting triggered.

Sample GitLab Configuration file with AppViewX SIGN+ CSP and Microsoft Signtool

```
.
.
.
job code_signing
 script:
    signtool.exe sign /f Codesign.cer  /fd sha256  /csp "AppViewX Enhanced Cryptographic Service Provider" /k "FF6CAB70-49EF-4A04-9ED6-967135E937E4" /tr
 "http://timestamp.digicert.com" /td sha265 <Path of Input Artifact>
.
.
.
```

> **Note:** The above script is an example showcasing the signing of an artifact generated post the build process using Microsoft Signtool and AppViewX CSP. The same can be extended to include the signing of other artifacts generated post build with tools like Nuget, Jarsigner, JSign etc.. using the commands generated in the README after executing the SIGN+ Installer executable in the GitLab Runner or any CI/CD Server.

Code Signing Integration with AppViewX CSP/PKCS#11

Using Signtool with AppViewX CSP

1. Run the AppViewX SIGN+ Installer executable to install the prerequisites required to use the AppViewX CSP/PKCS11 Providers.
2. Copy the signtool command generated in the README File and update the GitLab Configuration File with the appropriate script.

```
job code_signing
  stage: Sign using Signtool and AppViewX CSP
  script:
- signtool.exe sign /f <path to certificate>  /fd <digest algorithm>  /csp <csp_name> /k <key_alias_name> /tr <timestamp_url> /td <timestamp digest
 algorithm>  <input_file_path>
```

- **/f <path to certificate>:** Path to your code-signing certificate.
- **/fd <digest algorithm>:** Specifies the hashing algorithm.
- **/csp <csp_name>:** Name of Cryptographic Service Provider (CSP).
- **/k <key_alias_name>:** Key Container Name.
- **/tr <timestamp_url>:** Provides a timestamp from a trusted timestamping authority.
- **/tr <timestamp_digest>:** Specifies the timestamping Digest algorithm.
- **<input_file_path>:** Path to the file to be signed.

The **<path to certificate>**, **<digest algorithm>**, **<csp_name>**, **<key_alias_name>**, **<timestamp_url>**, **<timestamp_digest>** parametersare auto generated based on the signing policy configurations in the README after running the SIGN+ Installer.

Using JarSigner with AppViewX CSP

1. Run the AppViewX SIGN+ Installer executable to install the prerequisites required to use the AppViewX CSP/PKCS11 Providers.
2. Copy the Jarsigner command generated in the README File and update the GitLab Configuration File with the appropriate script.

```
job code_signing
  stage: Sign using Jarsigner and AppViewX CSP
  script:
    - jarsigner.exe -verbose -storetype "Windows-My" -keyStore NONE -tsa <time_stamp_url> <input_file_path> -signedjar <output_file_path> -sigalg
 <signature algorithm> <keypair alias>
```

The **<time_stamp_url>**, **<signature algorithm>** and **<keypair alias>** parameters are auto generated in the README after running the SIGN+ Installer.

Using Nuget with AppViewX CSP

1. Run the AppViewX SIGN+ Installer executable to install the prerequisites required to use the AppViewX CSP/PKCS11 Providers.
2. Copy the nuget command generated in the README File and update the GitLab Configuration File with the appropriate script.

```
job code_signing
  stage: Sign using Nuget and AppViewX CSP
  script:
   - nuget.exe sign <input_file_path> -Timestamper <timestamp_url> -CertificateFingerprint <certificate_fingerprint> -HashAlgorithm <hashing_algorithm>
-Verbosity detailed -Overwrite
```

The **<time_stamp_url>**, **<certificate_fingerprint>** and **<hashing_algorithm>** parameters are auto generated in the README after running the SIGN+ Installer.

Using JarSigner with AppViewX PKCS#11 Provider

1. Run the AppViewX SIGN+ Installer executable to install the prerequisites required to use the AppViewX CSP/PKCS11 Providers.
2. Copy the Jarsigner command generated in the README File and update the GitLab Configuration File with the appropriate script.

```
job code_signing
  stage: Sign using Jarsigner and AppViewX PKCS#11 Provider
  script:
   - jarsigner.exe -verbose -keystore NONE -storetype PKCS11 -certs -providerclass sun.security.pkcs11.SunPKCS11 -providerArg <path to
AVXPKCS11V1.cfg> <input_file_path> -signedjar <output_file_path> -tsa <time_stamp_url> -sigalg <signature algorithm> <keypairalias>
```

The **<path to AVXPKCS11V1.cfg>**, **<time_stamp_url>**, **<signature algorithm>** and **<keypair alias>** parameters are auto generated in the README after running the SIGN+ Installer.

Using JSign with AppViewX PKCS#11 Provider

1. Run the AppViewX SIGN+ Installer executable to install the prerequisites required to use the AppViewX CSP/PKCS11 Providers.
2. Copy the JSign command generated in the README File and update the GitLab Configuration File with the script.

```
job code_signing
  stage: Sign using JSign and AppViewX PKCS#11 Provider
  script:
   -  java -jar <path_to_jsign_jar> --keystore <path to AVXPKCS11V1.cfg> --storetype PKCS11 --storepass 12345678 --alias <keypair alias>  --alg <digest
algorithm> --tsaurl <timestamp url> <input_file_path>
```

The **<path to AVXPKCS11V1.cfg>**, **<keypair alias>**, **<digest algorithm>**, **<timestamp url>** parameters are auto generated based on the signing policy configurations in the README after running the SIGN+ Installer.

Using APKSigner with AppViewX PKCS#11 Provider

1. Run the AppViewX SIGN+ Installer executable to install the prerequisites required to use the AppViewX CSP/PKCS11 Providers.
2. Copy the APKSigner command generated in the README File and update the GitLab Configuration File with the appropriate script.

```
job code_signing
  stage: Sign using APKSigner and AppViewX PKCS#11 Provider
  script:
   - java -jar <path_to_apk_signer_jar> sign --provider-class sun.security.pkcs11.SunPKCS11 --provider-arg <path to AVXPKCS11V1.cfg> --ks NONE
--ks-type PKCS11 --ks-pass pass:12345678 --ks-key-alias <keypair alias> --in "<input_file_path>" --out "<output_file_path>" --v1-signing-enabled false
--v2-signing-enabled false --v3-signing-enabled true --v4-signing-enabled false
```

The **<path to AVXPKCS11V1.cfg>**, **<keypair alias>** parameters are auto generated based on the signing policy configurations in the README after running the SIGN+ Installer.

Integrating Code Signing in Azure Devops Pipeline

Azure Devops

Azure DevOps is a Software as a Service (SaaS) platform that provides tools for improved team collaboration. It also offers tools for automated build processes, version control, project management, testing, release management, package management, and more.

Azure Devops Pipeline

Azure DevOps Pipeline is a feature of Azure DevOps that allows you to create and run continuous integration and delivery (CI/CD) pipelines for applications. You can use Azure DevOps Pipeline to build, test, and deploy code to any platform and cloud.

Azure DevOps Configuration File

An Azure pipeline configuration file is a YAML file that defines the steps and tasks of your pipeline. YAML, which stands for "YAML Ain't Markup Language," is a human-readable data serialization language. A YAML file uses indentation and keywords to represent the structure and logic of your pipeline.

Here are some common elements in an Azure pipeline configuration file:

- **trigger:** A trigger defines the events that cause the pipeline to run. It can use branch names, tags, paths, or schedules to specify the trigger conditions.
- **pool:** A pool specifies the agent that runs the pipeline. An agent is a computing environment that executes one job at a time. It can use either Microsoft-hosted agents or self-hosted agents.
- **steps:** A step is the smallest building block of a pipeline. A step can run a command, tool, or task. A command is a shell command or a script. A tool is an executable file or a package. A task is a pre-packaged script that performs a specific action, such as publishing a build artifact or deploying to an environment.
- **inputs:** Inputs are parameters that control the behavior of a step. Inputs can be used to customize the configuration and options of a step, such as specifying the source folder, target folder, file pattern, or variable name.
- **displayName:** A display name is a user-friendly name that appears in the pipeline logs and UI, making your pipeline more readable and understandable.

- Code Signing Integration with Native Tools Using AppViewX SIGN+ in Azure DevOps
- Code Signing Integration with AppViewX CSP/PKCS#11

Code Signing Integration with Native Tools Using AppViewX SIGN+ in Azure DevOps

Prerequisites

1. Set up the Azure DevOps pipeline on the CI/CD build server to generate the required artifacts.
2. Download the **SIGN+_Package.zip** file and install it in the required build server and ensure connectivity from the build server to the SIGN+ Compute Cluster Node.

Sample Azure DevOps configuration file with AppViewX SIGN+ CSP

```
- task: PublishBuildArtifacts@1
  displayName: "Publish Unsigned DLL"
  inputs:
    PathtoPublish: '$(Build.ArtifactStagingDirectory)\Hello-World-Dot-Net.dll'
    ArtifactName: 'Unsigned DLL'
    publishLocation: 'Container'

- script: signtool.exe sign /f Codesign.cer  /fd sha256  /csp “AppViewX Enhanced Cryptographic Service Provider” /k
 "FF6CAB70-49EF-4A04-9ED6-967135E937E4" /tr “http://timestamp.digicert.com” /td sha256 $(Build.ArtifactStagingDirectory)\Hello-World-Dot-Net.exet
  displayName: "Sign Exe using AppViewX SIGN+"

- script: signtool.exe sign /f Codesign.cer  /fd sha256  /csp “AppViewX Enhanced Cryptographic Service Provider” /k
 "FF6CAB70-49EF-4A04-9ED6-967135E937E4" /tr “http://timestamp.digicert.com” /td sha256 $(Build.ArtifactStagingDirectory)\Hello-World-Dot-Net.dll
  displayName: "Sign DLL using AppViewX SIGN+"

- task: PublishBuildArtifacts@1
  displayName: "Publish Signed Exe"
  inputs:
    PathtoPublish: '$(Build.ArtifactStagingDirectory)\Hello-World-Dot-Net.exe'
    ArtifactName: 'Signed Exe'
    publishLocation: 'Container'


- task: PublishBuildArtifacts@1
  displayName: "Publish Signed DLL"
  inputs:
    PathtoPublish: '$(Build.ArtifactStagingDirectory)\Hello-World-Dot-Net.dll'
    ArtifactName: 'Signed DLL'
```

```
  publishLocation: 'Container'


- script: signtool verify /v /pa $(Build.ArtifactStagingDirectory)\Hello-World-Dot-Net.exe
  displayName: "Verify Exe"


- script: signtool verify /v /pa $(Build.ArtifactStagingDirectory)\Hello-World-Dot-Net.dll
  displayName: "Verify DLL"
```

Code Signing Integration with AppViewX CSP/PKCS#11

Using Signtool with AppViewX CSP

1. Execute the AppViewX SIGN+ Installer to set up the necessary prerequisites for utilizing the AppViewX CSP/PKCS11 Providers.
2. Copy the `signtool` command from the README file and incorporate it into the Azure Pipeline Configuration File by updating the relevant stage and script.

```
- script: signtool.exe sign /f <path to certificate>  /fd <digest algorithm>  /csp <csp_name> /k <key_alias_name> /tr <timestamp_url> /td <timestamp digest
 algorithm>  <input_file_path>
displayName: Signtool Signing
```

   - **/f <path to certificate>:** Path to your code-signing certificate.
   - **/fd <digest algorithm>:** Specifies the hashing algorithm.
   - **/csp <csp_name>:** Name of Cryptographic Service Provider (CSP).
   - **/k <key_alias_name>:** Key Container Name.
   - **/tr <timestamp_url>:** Provides a timestamp from a trusted timestamping authority.
   - **/tr <timestamp_digest>:** Specifies the timestamping Digest algorithm.
   - **<input_file_path>:** Path to the file to be signed.

   The parameters **<path to certificate>**, **<digest algorithm>**, **<csp_name>**, **<key_alias_name>**, **<timestamp_url>**, and **<timestamp_digest>** are automatically generated according to the signing policy configurations outlined in the README file after executing the SIGN+ Installer.

Using JarSigner with AppViewX CSP

1. Execute the AppViewX SIGN+ Installer to install the prerequisites for using the AppViewX CSP/PKCS11 Providers.
2. Copy the `jarsigner` command from the README file and update the Azure Pipeline Configuration File with the correct stage and script.

```
- script: jarsigner.exe -verbose -storetype "Windows-My" -keyStore NONE -tsa <time_stamp_url> <input_file_path> -signedjar <output_file_path> -sigalg
 <signature algorithm> <keypair alias>
 displayName: Jarsigner Signing
```

The parameters **<time_stamp_url>**, **<signature algorithm>** and **<keypair alias>** are automatically generated in the README file after executing the SIGN+ Installer.

Using Nuget with AppViewX CSP

1. Execute the AppViewX SIGN+ Installer to set up the prerequisites for using the AppViewX CSP/ PKCS11 Providers.
2. Copy the `nuget` command from the README file and update the Azure Pipeline Configuration File with the relevant stage and script.

```
- script: nuget.exe sign <input_file_path> -Timestamper <timestamp_url> -CertificateFingerprint <certificate_fingerprint> -HashAlgorithm
 <hashing_algorithm> -Verbosity detailed -Overwrite
 displayName: Nuget Signing
```

The parameters **<time_stamp_url>**, **<certificate_fingerprint>** and **<hashing_algorithm>** are automatically generated in the README file after executing the SIGN+ Installer.

Using JarSigner with AppViewX PKCS#11 Provider

1. Execute the AppViewX SIGN+ Installer to install the prerequisites needed for the AppViewX CSP/ PKCS11 Providers.
2. Copy the `jarsigner` command from the README file and update the Azure Pipeline Configuration File with the corresponding stage and script.

```
- script: jarsigner.exe -verbose -keystore NONE -storetype PKCS11 -certs -providerclass sun.security.pkcs11.SunPKCS11 -providerArg <path to
 AVXPKCS11V1.cfg> <input_file_path> -signedjar <output_file_path> -tsa <time_stamp_url> -sigalg <signature algorithm> <keypairalias>
 displayName: Jarsigner Signing
```

The parameters **<path to AVXPKCS11V1.cfg>**, **<time_stamp_url>**, **<signature algorithm>** and **<keypair alias>** are automatically generated in the README file after executing the SIGN+ Installer.

Using JSign with AppViewX PKCS#11 Provider

1. Execute the AppViewX SIGN+ Installer to install the prerequisites necessary for using the AppViewX CSP/PKCS11 Providers.
2. Copy the `JSign` command from the README file and update the Azure Pipeline Configuration File with the appropriate stage and script.

```
- script: java -jar <path_to_jsign_jar> --keystore <path to AVXPKCS11V1.cfg> --storetype PKCS11 --storepass 12345678 --alias <keypair alias>  --alg
 <digest algorithm> --tsaurl <timestamp url> <input_file_path>
 displayName: JSign Signing
```

The parameters **<path to AVXPKCS11V1.cfg>**, **<keypair alias>**, **<digest algorithm>** and **<timestamp url>** are automatically generated according to the signing policy configurations outlined in the README file after executing the SIGN+ Installer.

Using APKSigner with AppViewX PKCS#11 Provider

1. Run the AppViewX SIGN+ Installer to install the prerequisites for using the AppViewX CSP/PKCS11 Providers.
2. Copy the `APKSigner` command from the README file and update the Azure Pipeline Configuration File with the corresponding stage and script.

```
- script: java -jar <path_to_apk_signer_jar> sign --provider-class sun.security.pkcs11.SunPKCS11 --provider-arg <path to AVXPKCS11V1.cfg> --ks NONE
 --ks-type PKCS11 --ks-pass pass:12345678 --ks-key-alias <keypair alias> --in "<input_file_path>" --out "<output_file_path>" --v1-signing-enabled false
 --v2-signing-enabled false --v3-signing-enabled true --v4-signing-enabled false
 displayName: APKSigner Signing
```

The parameters **<path to AVXPKCS11V1.cfg>**, **<keypair alias>** are automatically generated according to the signing policy configurations outlined in the README file after executing the SIGN+ Installer.

Integrating Code Signing in GitHub Actions Pipeline

GitHub Actions Pipeline

GitHub Actions is a continuous integration and continuous development (CI/CD) platform that allows users to automate their build, test, and deployment pipeline. Users may design workflows that build and test every pull and push request to their repository or deploy merged pull requests to production. GitHub Actions is a powerful tool that allows developers to automate workflows within their GitHub repositories. Each workflow is made up of one or more jobs, which are made up of one or more steps. Each step is a set of commands that are executed on a runner, which is a virtual machine that runs the required workflows.

GitHub Actions Configuration File

To get started with GitHub Actions, a GitHub account and a repository is required. Once creating the repository, create a new workflow by adding a YAML file to the ***.github/workflows*** directory in the created repository. Some of the terms used in the YAML file defining workflow are as follows:

- **name:** name of the workflow.
- **on:** specifies when the workflow should be triggered. For example, the workflow can run when a pull request is opened on a branch.
- **jobs:** a list of jobs that will be executed as part of the workflow.
- **run-on:** specifies the operating system and environment for the job.
- **steps:** specifies a list of steps that will be executed as part of the job.
- **uses:** a shortcut for using an existing action from the GitHub Marketplace.
- **name:** specifies the name of the step.
- **run:** This is a shell command that will be executed as part of the step.

Code Signing Integration with Native Tools using AppViewX SIGN+ in Github Actions Pipeline:

Prerequisites

1. A GitHub repository with GitHub Actions pipeline setup in the runner.
2. Download the SIGN+_Package.zip for the required OS and install in the required build server/runner and ensure connectivity from the build server/runner to the SIGN+ API Connector URL.

Sample Github Actions Configuration file with AppViewX SIGN+ CSP and Microsoft Signtool

```
name: Code Signing using AppViewX SIGN+


on:
  push:
    branches: <[ Branch Name ]>


jobs:
  build:
    runs-on: <Runner name>
    steps:
      - name: Checkout code
```

```
    uses: actions/checkout@v2


  - name: Build code
    uses: msbuild <build parameters>


  - name: Sign code using AppViewX CSP
    run: |
     signtool.exe sign /f Codesign.cer  /fd sha256  /csp "AppViewX Enhanced Cryptographic Service Provider" /k
"FF6CAB70-49EF-4A04-9ED6-967135E937E4" /tr "http://timestamp.digicert.com" /td sha265 <Path of Input Artifact>
```

Note: The above script is an example showcasing the signing of an artifact generated post the build process using Microsoft Signtool and AppViewX CSP. The same can be extended to include the signing of other artifacts generated post build with tools like Nuget, Jarsigner, JSign etc.. using the commands generated in the README after executing the SIGN+ Installer executable in the GitHub Actions workflow or any CI/CD Server.

Code Signing Integration with AppViewX CSP/PKCS#11:

Using Signtool with AppViewX CSP

1. Run the AppViewX SIGN+ Installer executable to install the prerequisites required to use the AppViewX CSP/PKCS11 Providers.
2. Copy the signtool command generated in the README File and update the Github Actions Configuration File with the appropriate script.

```
- name: Sign using Signtool and AppViewX CSP
- script: signtool.exe sign /f <path to certificate>  /fd <digest algorithm>  /csp <csp_name> /k <key_alias_name> /tr <timestamp_url> /td <timestamp digest
 algorithm>  <input_file_path>
```

   - **/f <path to certificate>:** Path to your code-signing certificate.
   - **/fd <digest algorithm>:** Specifies the hashing algorithm.
   - **/csp <csp_name>:** Name of Cryptographic Service Provider (CSP).
   - **/k <key_alias_name>:** Key Container Name.
   - **/tr <timestamp_url>:** Provides a timestamp from a trusted timestamping authority.
   - **/tr <timestamp_digest>:** Specifies the timestamping Digest algorithm.
   - **<input_file_path>:** Path to the file to be signed.

   The **<path to certificate>, <digest algorithm>, <csp_name>, <key_alias_name>, <timestamp_url>, <timestamp_digest>** parameters are auto generated based on the signing policy configurations in the README after running the SIGN+ Installer.

Using JarSigner with AppViewX CSP

1. Run the AppViewX SIGN+ Installer executable to install the prerequisites required to use the AppViewX CSP/PKCS11 Providers.
2. Copy the Jarsigner command generated in the README File and update the Github Actions Configuration File with the appropriate script.

```
- name: Sign using Jarsigner and AppViewX CSP
- script: jarsigner.exe -verbose -storetype "Windows-My" -keyStore NONE -tsa <time_stamp_url> <input_file_path> -signedjar <output_file_path> -sigalg
 <signature algorithm> <keypair alias>
```

The **<time_stamp_url>, <signature algorithm>** and **<keypair alias>** parameters are auto generated in the README after running the SIGN+ Installer.

Using Nuget with AppViewX CSP

1. Run the AppViewX SIGN+ Installer executable to install the prerequisites required to use the AppViewX CSP/PKCS11 Providers.
2. Copy the nuget command generated in the README File and update the Github Actions Configuration File with the appropriate script.

```
- name: Sign using Nuget and AppViewX CSP
- script: nuget.exe sign <input_file_path> -Timestamper <timestamp_url> -CertificateFingerprint <certificate_fingerprint> -HashAlgorithm
 <hashing_algorithm> -Verbosity detailed -Overwrite
```

The **<time_stamp_url>, <certificate_fingerprint>** and **<hashing_algorithm>** parameters are auto generated in the README after running the SIGN+ Installer.

Using JarSigner with AppViewX PKCS#11 Provider

1. Run the AppViewX SIGN+ Installer executable to install the prerequisites required to use the AppViewX CSP/PKCS11 Providers.
2. Copy the Jarsigner command generated in the README File and update the Github Actions Configuration File with the appropriate script.

```
- name: Sign using Jarsigner and AppViewX PKCS#11 Provider
- script: jarsigner.exe -verbose -keystore NONE -storetype PKCS11 -certs -providerclass sun.security.pkcs11.SunPKCS11 -providerArg <path to
 AVXPKCS11V1.cfg> <input_file_path> -signedjar <output_file_path> -tsa <time_stamp_url> -sigalg <signature algorithm> <keypairalias>
```

The **<path to AVXPKCS11V1.cfg>, <time_stamp_url>, <signature algorithm> and <keypair alias>** parameters are auto generated in the README after running the SIGN+ Installer.

Using JSign with AppViewX PKCS#11 Provider

1. Run the AppViewX SIGN+ Installer executable to install the prerequisites required to use the AppViewX CSP/PKCS11 Providers.
2. Copy the JSign command generated in the README File and update the Github Actions Configuration File with the script.

```
- name: Sign using JSign and AppViewX PKCS#11 Provider

- script: java -jar <path_to_jsign_jar> --keystore <path to AVXPKCS11V1.cfg> --storetype PKCS11 --storepass 12345678 --alias <keypair alias>  --alg

 <digest algorithm> --tsaurl <timestamp url> <input_file_path>
```

The **<path to AVXPKCS11V1.cfg>, <keypair alias>, <digest algorithm>, <timestamp url>** parameters are auto generated based on the signing policy configurations in the README after running the SIGN+ Installer.

Using APKSigner with AppViewX PKCS#11 Provider

1. Run the AppViewX SIGN+ Installer executable to install the prerequisites required to use the AppViewX CSP/PKCS11 Providers.
2. Copy the APKSigner command generated in the README File and update the Github Actions Configuration File with the appropriate script.

```
- name: Sign using APKSigner and AppViewX PKCS#11 Provider

- script: java -jar <path_to_apk_signer_jar> sign --provider-class sun.security.pkcs11.SunPKCS11 --provider-arg <path to AVXPKCS11V1.cfg> --ks NONE

 --ks-type PKCS11 --ks-pass pass:12345678 --ks-key-alias <keypair alias> --in "<input_file_path>" --out "<output_file_path>" --v1-signing-enabled false

 --v2-signing-enabled false --v3-signing-enabled true --v4-signing-enabled false
```

The **<path to AVXPKCS11V1.cfg>**, **<keypair alias>** parameters are auto generated based on the signing policy configurations in the README after running the SIGN+ Installer.

Integrating Code Signing in Atlassian Bamboo Pipeline

Bamboo

Bamboo is a continuous integration and continuous deployment (CI/CD) server that is used to automate the build, test, and deployment processes of software applications. Bamboo is designed to streamline and automate the software development process, helping teams deliver software more efficiently and with fewer errors. Bamboo assists software development teams by providing:

- Automated building and testing of software source-code status
- Updates on successful and failed builds
- Reporting tools for statistical analysis.

Bamboo Pipeline

The Bamboo pipeline refers to a sequence of automated steps and tasks that are defined to build, test, and deploy a software application as part of a continuous integration and continuous deployment (CI/CD) process.

Here are the key components and concepts associated with Bamboo pipelines:

1. **Stages:** A pipeline is typically divided into stages, each representing a phase in the CI/CD process. Common stages include "Build," "Test," "Deploy to Staging," and "Deploy to Production." Stages are executed sequentially, and each stage may consist of one or more jobs.
2. **Jobs:** Within each stage, you define one or more jobs. A job is a collection of tasks that need to be executed together. For example, a "Build" stage might have a single job that compiles source code, runs unit tests, and packages the application. Bamboo provides a wide range of built-in tasks and supports custom scripts and commands.
3. **Tasks:** Tasks are individual steps within a job that perform specific actions, such as running a script, checking out code from a version control system, or publishing artifacts. Bamboo offers a variety of task types to accommodate different actions.
4. **Triggers:** Pipelines can be triggered manually or automatically based on events. Automatic triggers can be set up to start a pipeline when code is pushed to a version control repository, ensuring that new changes are continuously integrated and tested.
5. **Branches:** Bamboo pipelines can be configured to work with different branches of your version control repository. This allows you to have separate CI/CD pipelines for different development branches, such as feature branches or release branches.
6. **Artifacts:** Bamboo allows you to manage and store build artifacts generated during the pipeline, making it easy to distribute them to different environments or store them for future reference.
7. **Notifications:** Bamboo can send notifications and reports about pipeline results to team members via email, chat, or other communication channels.
8. **Parallel Execution:** Bamboo supports parallel execution of tasks and jobs within a stage, enabling faster build and test times by taking advantage of available resources.

For more information on configuring the Bamboo CI Server, Understanding the Bamboo CI Server.

Bamboo Configuration File

Bamboo's configuration is primarily managed through its web-based interface, and it doesn't typically rely on single configuration files. Bamboo relies on a distributed configuration model where various configuration settings are stored in different places and files, and many of these settings are managed through its web-based administration interface. The primary configuration files in Bamboo are associated with the Bamboo home directory (BAMBOO_HOME) and Bamboo agents.

Some of the key configuration files and directories in Bamboo are:

1. **Bamboo Home Directory (BAMBOO_HOME):** This directory contains many of the configuration files and data for Bamboo. The specific location and structure may vary based on your installation. Important subdirectories and files include:
   • xml-data/: Contains various XML configuration files
   • lib/: May include libraries and JAR files for custom plugins and extensions
   • agent/: Configuration files and data specific to Bamboo agents.
2. **Bamboo Agent Configuration:** Each Bamboo agent has its own configuration file named “bamboo-agent.cfg.xml”. This file contains agent-specific settings, including the Bamboo server connection details.

   Sample bamboo-agent.cfg.xml:

```
<?xml version="1.0" encoding="UTF-8"?>


<bamboo-agent>
    <!-- Bamboo server URL -->
    <serverUrl>http://bamboo-server:8085</serverUrl>


    <!-- Bamboo agent's unique identifier -->
    <agentUuid>YOUR_AGENT_UUID</agentUuid>


    <!-- Bamboo agent's display name -->
    <agentName>My Bamboo Agent</agentName>


    <!-- Bamboo agent's capabilities -->
    <capabilities>
        <!-- Example capability for Maven -->
        <capability name="system.builder.mvn3.Maven 3" value="/usr/local/apache-maven-3.8.1" />
```

```
    <!-- Example capability for Node.js -->
    <capability name="system.builder.node.Node.js" value="/usr/local/bin/node" />
  </capabilities>

  <!-- Bamboo agent's working directory -->
  <workingDir>/path/to/agent/work</workingDir>

  <!-- Bamboo agent's temp directory -->
  <tempDir>/path/to/agent/temp</tempDir>

  <!-- Bamboo agent's home directory -->
  <homeDir>/path/to/agent/home</homeDir>

  <!-- Bamboo agent's capabilities sharing method (usually "true" or "false") -->
  <sharingCapability>true</sharingCapability>

  <!-- Bamboo agent's environment variables -->
  <environmentVariables>
    <environmentVariable name="PATH" value="/usr/local/bin:/usr/bin:/bin" />
    <environmentVariable name="JAVA_HOME" value="/usr/local/jdk1.8.0_291" />
  </environmentVariables>

  <!-- Bamboo agent's security token (if required) -->
  <securityToken>YOUR_SECURITY_TOKEN</securityToken>
</bamboo-agent>
```

Configuring the Bamboo Pipeline Environment

1. **Create or Open a Bamboo Plan:**
   - Login to the Bamboo instance and create a new plan or open an existing one where the code signing has to be integrated.
2. **Add a Script Task:**
   - Within the Bamboo plan, add a new Script task to one of the existing jobs or create a new job for this purpose. This task will run the jarsigner command.
   - Set the interpreter to "**Windows PowerShell**" if using a Windows agent or set the interpreter to "**Shell**" if using a Linux agent.
3. **Configure the Script Task for Integration with AppViewX CSP/PKCS#11:**

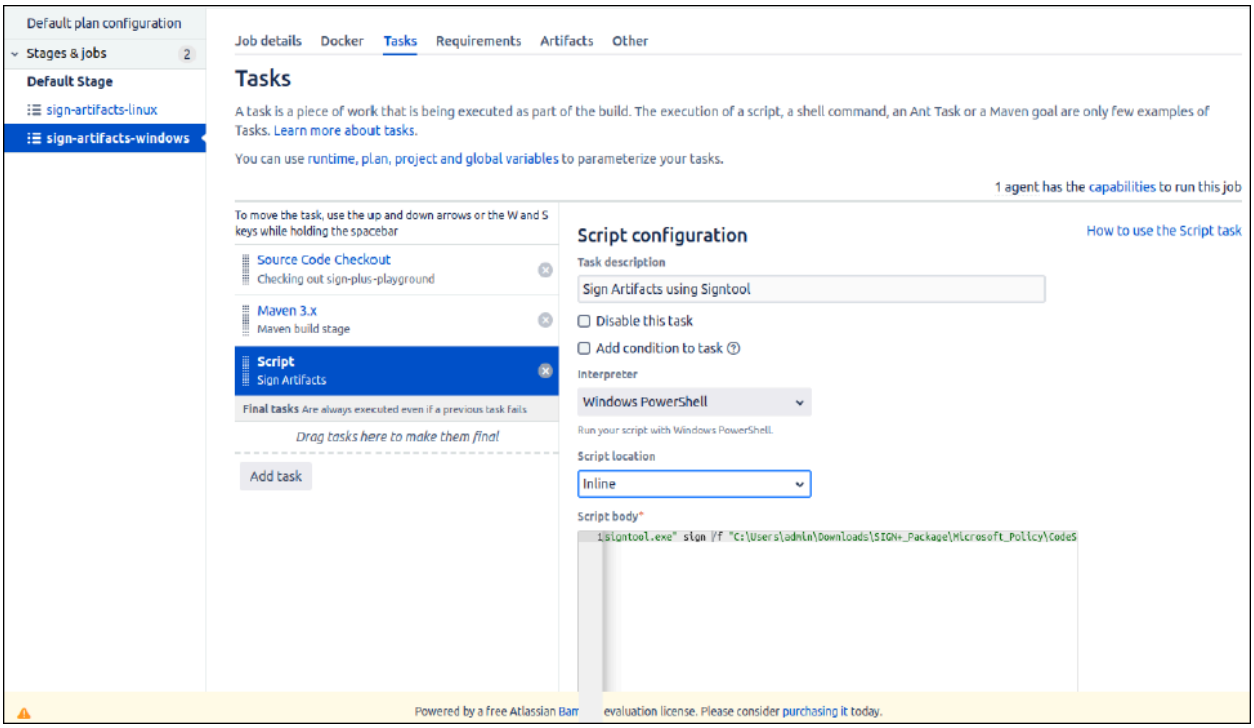
In the Script Body section of the Script task configuration, add the required commands to sign the artifacts based on requirement.

4. **Save and Execute:**
   - Save the Script task configuration.
   - Trigger a Bamboo build for the plan or setup webhooks to trigger the task based on code commit or any other events based on configuration.

Prerequisites

1. The pipeline should be configured with the required Build stages and the required artifacts should be ready for signing.
2. Copied the downloaded SIGN+_Package to the configured runner machine or agent and installed the package.
3. Ensure the connectivity from the runner machine to the SIGN+ API Connector URL Node (Compute Cluster, Cloud Connector, LoadBalancer or OnPrem Worker Node).

Sample Script Configuration using AppViewX CSP and Signtool in Bamboo Dashboard



Using Signtool with AppViewX CSP

1. Execute the AppViewX SIGN+ Installer executable in the configured runner machine to install the prerequisites required to use the AppViewX CSP/PKCS11 Providers.
2. Copy the `signtool` command generated in the README File after installation and update the Script Section with the generated command.

```
signtool.exe sign /f <path to certificate> /fd <digest algorithm> /csp <csp_name> /k <key_alias_name>
/tr <timestamp_url> /td <timestamp digest algorithm> <input_file_path>
```

   - **/f <path to certificate>:** Path to your code-signing certificate.
   - **/fd <digest algorithm>:** Specifies the hashing algorithm.
   - **/csp <csp_name>:** Name of Cryptographic Service Provider (CSP).
   - **/k <key_alias_name>:** Key Container Name.
   - **/tr <timestamp_url>:** Provides a timestamp from a trusted timestamping authority.
   - **/tr <timestamp_digest>:** Specifies the timestamping Digest algorithm.
   - **<input_file_path>:** Path to the file to be signed.

   The **<path to certificate>**, **<digest algorithm>**, **<csp_name>**, **<key_alias_name>**, **<timestamp_url>**, **<timestamp_digest>** parameters are auto generated based on the signing policy configurations in the README after running the SIGN+ Installer.

Using JarSigner with AppViewX CSP

1. Execute the AppViewX SIGN+ Installer executable in the configured runner machine to install the prerequisites required to use the AppViewX CSP/PKCS11 Providers.
2. Copy the `Jarsigner` command generated in the README File and update the Script Section with the generated command.

```
jarsigner.exe -verbose -storetype "Windows-My" -keyStore NONE -tsa <time_stamp_url>
<input_file_path> -signedjar <output_file_path> -sigalg <signature algorithm> <keypair alias>
```

   The **<time_stamp_url>**, **<signature algorithm>** and **<keypair alias>** parameters are auto generated in the README after running the SIGN+ Installer.

Using Nuget with AppViewX CSP

1. Execute the AppViewX SIGN+ Installer executable in the configured runner machine to install the prerequisites required to use the AppViewX CSP/PKCS11 Providers.
2. Copy the `nuget` command generated in the README File and update the Script Section with the generated command.

```
nuget.exe sign <input_file_path> -Timestamper <timestamp_url> -CertificateFingerprint
<certificate_fingerprint> -HashAlgorithm <hashing_algorithm> -Verbosity detailed -Overwrite
```

The **<time_stamp_url>**, **<certificate_fingerprint>** and **<hashing_algorithm>** parameters are auto generated in the README after running the SIGN+ Installer.

Using JarSigner with AppViewX PKCS#11 Provider

1. Execute the AppViewX SIGN+ Installer executable in the configured runner machine to install the prerequisites required to use the AppViewX CSP/PKCS11 Providers.
2. Copy the `Jarsigner` command generated in the README File and update the Script Section with the generated command.

```
jarsigner -verbose -keystore NONE -storetype PKCS11 -certs -providerclass
sun.security.pkcs11.SunPKCS11 -providerArg <path to AVXPKCS11V1.cfg> <input_file_path> -signedjar
<output_file_path> -tsa <time_stamp_url> -sigalg <signature algorithm> <keypairalias>
```

The **<path to AVXPKCS11V1.cfg>**, **<time_stamp_url>**, **<signature algorithm>** and **<keypair alias>** parameters are auto generated in the README after running the SIGN+ Installer.

Using JSign with AppViewX PKCS#11 Provider

1. Execute the AppViewX SIGN+ Installer executable in the configured runner machine to install the prerequisites required to use the AppViewX CSP/PKCS11 Providers.
2. Copy the `JSign` command generated in the README File and update the Script Section with the generated command.

```
java -jar <path_to_jsign_jar> --keystore <path to AVXPKCS11V1.cfg> --storetype PKCS11 --storepass
12345678 --alias <keypair alias> --alg <digest algorithm> --tsaurl <timestamp url> <input_file_path>
```

The **<path to AVXPKCS11V1.cfg>**, **<keypair alias>**, **<digest algorithm>**, **<timestamp url>** parameters are auto generated based on the signing policy configurations in the README after running the SIGN+ Installer.

Using APKSigner with AppViewX PKCS#11 Provider

1. Execute the AppViewX SIGN+ Installer executable in the configured runner machine to install the prerequisites required to use the AppViewX CSP/PKCS11 Providers.
2. Copy the `APKSigner` command generated in the README File and update the Script Section with the generated command.

```
java -jar <path_to_apk_signer_jar> sign --provider-class sun.security.pkcs11.SunPKCS11 --provider-arg
<path to AVXPKCS11V1.cfg> --ks NONE --ks-type PKCS11 --ks-pass pass:12345678 --ks-key-alias
<keypair alias> --in "<input_file_path>" --out "<output_file_path>" --v1-signing-enabled false
--v2-signing-enabled false --v3-signing-enabled true --v4-signing-enabled false
```

The **<path to AVXPKCS11V1.cfg>**, **<keypair alias>** parameters are auto generated based on the signing policy configurations in the README after running the SIGN+ Installer.

Note: The script can be configured to sign with any tool using the commands generated in the README File based on requirement.

Integrating Code Signing in AWS DevOps

AWS CodePipeline

AWS CodePipeline is a continuous delivery service that enables you to model, visualize, and automate the steps required to release your software. With AWS CodePipeline, you model the full release process for building your code, deploying to pre-production environments, testing your application and releasing it to production. AWS CodePipeline then builds, tests, and deploys your application according to the defined workflow every time there is a code change. You can integrate partner tools and your own custom tools into any stage of the release process to form an end-to-end continuous delivery solution

Some of the key components and concepts associated with AWS CodePipeline are:

1. **Pipeline:** A pipeline is a workflow construct that describes how software changes go through a release process. Each pipeline is made up of a series of stages.
2. **Stages:** A stage is a logical unit you can use to isolate an environment and to limit the number of concurrent changes in that environment. Each stage contains actions that are performed on the application artifacts. Your source code is an example of an artifact. A stage might be a build stage, where the source code is built and tests are run. It can also be a deployment stage, where code is deployed to runtime environments. Each stage is made up of a series of serial or parallel actions.
3. **Transitions:** A transition is the point where a pipeline execution moves to the next stage in the pipeline. You can disable a stage's inbound transition to prevent executions from entering that stage, and then you can enable the transition to allow executions to continue. When more than one execution arrives at a disabled transition, only the latest execution continues to the next stage when the transition is enabled. This means that newer executions continue to supersede waiting executions while the transition is disabled, and then after the transition is enabled, the execution that continues is the superseding execution.

4. **Actions:** An action is a set of operations performed on application code and configured so that the actions run in the pipeline at a specified point. This can include things like a source action from a code change, an action for deploying the application to instances, and so on. For example, a deployment stage might contain a deployment action that deploys code to a compute service like Amazon EC2 or AWS Lambda.Valid CodePipeline action types are source, build, test, deploy, approval, and invoke.

For more information on the concepts of AWS CodePipeline, refer CodePipeline concepts.

CodePipeline Configuration File

Sample Github Actions Configuration file with AppViewX SIGN+ CSP and Jarsigner

```
version: 0.2


phases:
 install:
  runtime-versions:
    java: openjdk17
    maven: 3.8


 pre_build:
  commands:
    # Checkout code from CodeCommit (Note: AWS CodeBuild automatically clones the repository)
    - echo "Pre-build phase started"
    - echo "Source code checkout completed"
    - echo "running as $(whoami)"


 build:
  commands:
    # Maven build
    - echo "Working directory: $(pwd)"
    - echo "Build phase started"
    - mvn clean install


 post_build:
  commands:
    - echo "Post-build phase started"
    - echo "Working directory: $(pwd)"
    - echo "signing artifacts"
```

```
    - jarsigner.exe -verbose -keystore NONE -storetype PKCS11 -certs -providerclass sun.security.pkcs11.SunPKCS11 -providerArg
"C:\Windows\system32\config\systemprofile\AppData\Roaming\AppViewX Sign+\AVXPKCS11V1.cfg" -storepass NONE target/simple-poc-1.0.0.jar -signedjar
target/simple-poc-1.0.0_signed.jar -tsa "http://timestamp.digicert.com" -sigalg "SHA256withRSA" "AppViewX Inc Test's AppViewX Intermediate CA"
    - echo "Build completed successfully"


artifacts:
 files:
  - target/**/*
 discard-paths: no
```

For more information on declaring the pipeline configuration , refer [Pipeline declaration - AWS CodePipeline]

Note: The above script is an example showcasing the signing of an artifact generated post the build process using Jarsigner and AppViewX PKCS11. The same can be extended to include the signing of other artifacts generated post build with tools like Nuget, Jarsigner, JSign etc.. using the commands generated in the README after executing the SIGN+ Installer executable in the runner machine or CI/CD Server.

Code Signing Integration with Native Tools using AppViewX SIGN+ in AWS CodePipeline:

Prerequisites

1. The pipeline should be configured with the required Build stages and the required artifacts should be ready for signing.
2. Copied the downloaded SIGN+_Package to the configured runner machine or agent and installed the package.
3. Ensure the connectivity from the runner machine to the SIGN+ API Connector URL Node (Compute Cluster, Cloud Connector, LoadBalancer or OnPrem Worker Node).

Using Signtool with AppViewX CSP

1. Run the AppViewX SIGN+ Installer executable in the configured runner machine to install the prerequisites required to use the AppViewX CSP/PKCS11 Providers.
2. Copy the signtool command generated in the README File after installation and update the Script Section with the generated command:

```
signtool.exe sign /f <path to certificate>  /fd <digest algorithm>  /csp <csp_name> /k <key_alias_name> /tr <timestamp_url> /td <timestamp digest algorithm>
 <input_file_path>
```

- **/f <path to certificate>:** Path to your code-signing certificate.
- **/fd <digest algorithm>:** Specifies the hashing algorithm.
- **/csp <csp_name>:** Name of Cryptographic Service Provider (CSP).
- **/k <key_alias_name>:** Key Container Name.
- **/tr <timestamp_url>:** Provides a timestamp from a trusted timestamping authority.
- **/tr <timestamp_digest>:** Specifies the timestamping Digest algorithm.
- **<input_file_path>:** Path to the file to be signed.

The **<path to certificate>, <digest algorithm>, <csp_name>, <key_alias_name>, <timestamp_url>, <timestamp_digest>** parameters are auto generated based on the signing policy configurations in the README after running the SIGN+ Installer.

Using JarSigner with AppViewX CSP

1. Run the AppViewX SIGN+ Installer executable in the configured runner machine to install the prerequisites required to use the AppViewX CSP/PKCS11 Providers.
2. Copy the Jarsigner command generated in the README File and update the Script Section with the generated command:

```
jarsigner.exe -verbose -storetype "Windows-My" -keyStore NONE -tsa <time_stamp_url> <input_file_path> -signedjar <output_file_path> -sigalg <signature
 algorithm> <keypair alias>
```

The **<time_stamp_url>, <signature algorithm>** and **<keypair alias>** parameters are auto generated in the README after running the SIGN+ Installer.

Using Nuget with AppViewX CSP

1. Run the AppViewX SIGN+ Installer executable in the configured runner machine to install the prerequisites required to use the AppViewX CSP/PKCS11 Providers.
2. Copy the nuget command generated in the README File and update the Script Section with the generated command:

```
nuget.exe sign <input_file_path> -Timestamper <timestamp_url> -CertificateFingerprint <certificate_fingerprint> -HashAlgorithm <hashing_algorithm>
 -Verbosity detailed -Overwrite
```

The **<time_stamp_url>, <certificate_fingerprint>** and **<hashing_algorithm>** parameters are auto generated in the README after running the SIGN+ Installer.

Using JarSigner with AppViewX PKCS#11 Provider

1. Run the AppViewX SIGN+ Installer executable in the configured runner machine to install the prerequisites required to use the AppViewX CSP/PKCS11 Providers.
2. Copy the Jarsigner command generated in the README File and update the Script Section with the generated command:

```
jarsigner -verbose -keystore NONE -storetype PKCS11 -certs -providerclass sun.security.pkcs11.SunPKCS11 -providerArg <path to AVXPKCS11V1.cfg>
 <input_file_path> -signedjar <output_file_path> -tsa <time_stamp_url> -sigalg <signature algorithm> <keypairalias>
```

   The **<path to AVXPKCS11V1.cfg>, <time_stamp_url>, <signature algorithm>** and **<keypair alias>** parameters are auto generated in the README after running the SIGN+ Installer.

Using JSign with AppViewX PKCS#11 Provider

1. Run the AppViewX SIGN+ Installer executable in the configured runner machine to install the prerequisites required to use the AppViewX CSP/PKCS11 Providers.
2. Copy the JSign command generated in the README File and update the Script Section with the generated command:

```
java -jar <path_to_jsign_jar> --keystore <path to AVXPKCS11V1.cfg> --storetype PKCS11 --storepass 12345678 --alias <keypair alias>  --alg <digest
 algorithm> --tsaurl <timestamp url> <input_file_path>
```

   The **<path to AVXPKCS11V1.cfg>, <keypair alias>, <digest algorithm>, <timestamp url>** parameters are auto generated based on the signing policy configurations in the README after running the SIGN+ Installer.

Using APKSigner with AppViewX PKCS#11 Provider

1. Run the AppViewX SIGN+ Installer executable in the configured runner machine to install the prerequisites required to use the AppViewX CSP/PKCS11 Providers.
2. Copy the APKSigner command generated in the README File and update the Script Section with the generated command:

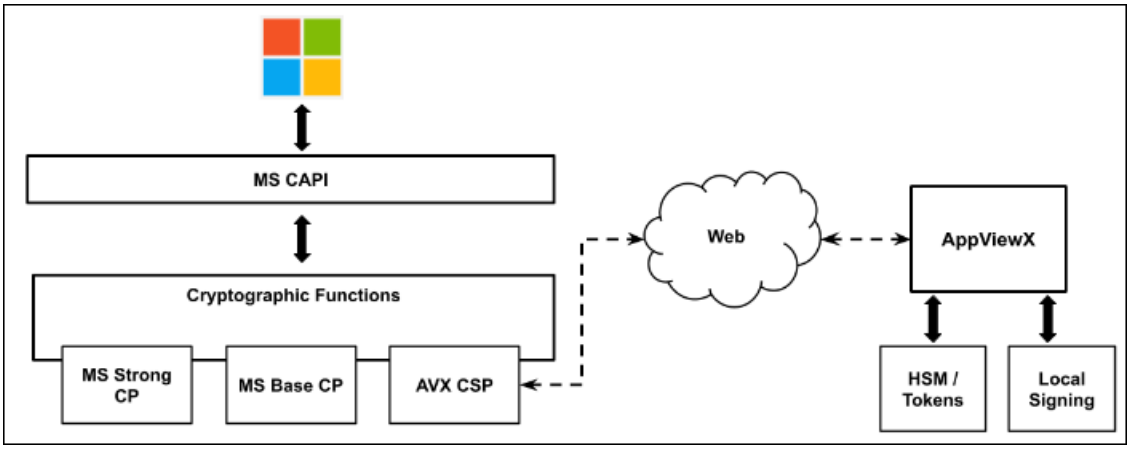
```
java -jar <path_to_apk_signer_jar> sign --provider-class sun.security.pkcs11.SunPKCS11 --provider-arg <path to AVXPKCS11V1.cfg> --ks NONE
 --ks-type PKCS11 --ks-pass pass:12345678 --ks-key-alias <keypair alias> --in "<input_file_path>" --out "<output_file_path>" --v1-signing-enabled false
 --v2-signing-enabled false --v3-signing-enabled true --v4-signing-enabled false
```

The **<path to AVXPKCS11V1.cfg>, <keypair alias>** parameters are auto generated based on the signing policy configurations in the README after running the SIGN+ Installer.

Note: The script can be configured to sign with any tool using the commands generated in the README File based on requirement.

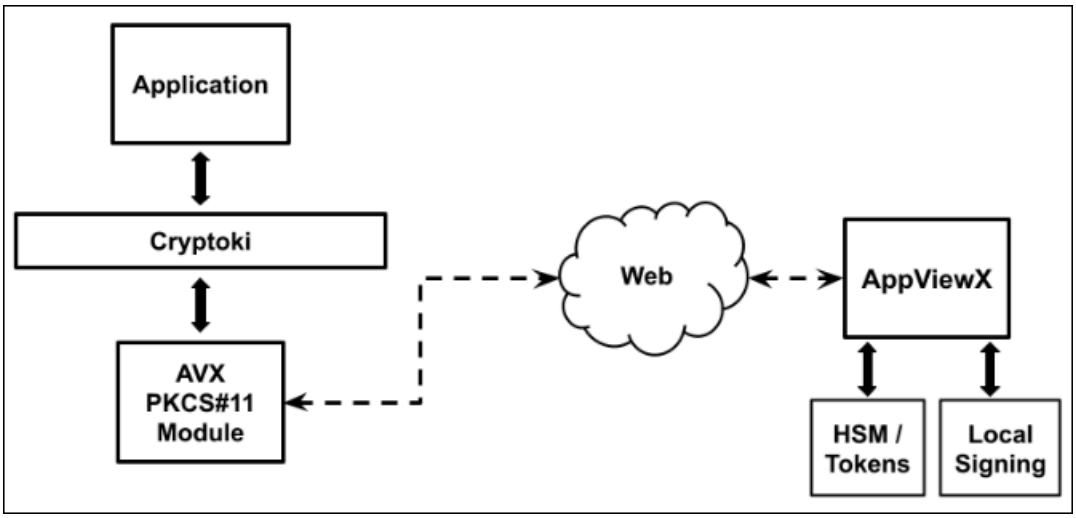
Appendix

AppViewX Cryptographic Service Provider (CSP) Working Flow



The AppViewX CSP offers built-in integration with Windows Crypto API function calls, allowing seamless compatibility with native signing tools such as Microsoft Signtool. This integration enables secure remote communication with AppViewX's centralized key management solution, facilitating efficient on-demand code signing processes.

AppViewX PKCS#11 Provider Working Flow



The AppViewX PKCS11 Provider seamlessly integrates with native PKCS11 tools, ensuring compatibility and interoperability. This integration enables secure and efficient communication with AppViewX's centralized key management solution, facilitating on-demand code signing processes effectively.

Integration with IDE

SIGN+ Integration with IDE refers to the seamless connection between a software application or tool and an Integrated Development Environment (IDE). This integration enhances the development workflow by allowing developers to access, manage, and interact with tools and features directly within their IDE.

- Integrating Code Signing in InstallShield
- Integrating Code Signing using Scripts
- ClickOnce Deployment with Visual Studio

Integrating Code Signing in InstallShield

Installshield

InstallShield is a proprietary software tool used for creating installers or software packages. It is primarily designed for installing software on Microsoft Windows desktop and server platforms. Additionally, it can manage software applications and packages across various handheld and mobile devices.

Download Installshield

1. Download from InstallShield Windows Installer Get Your Free Trial Today | Revenera.
2. Install using the Installation Wizard.

• Sign Installer files with Installshield using AppViewX CSP

• Troubleshoot Signing Errors

Sign Installer files with Installshield using AppViewX CSP

InstallShield allows users to create flexible installations quickly and easily across all Windows operating systems. It is easy for development teams to be more agile, flexible and collaborative when building reliable Windows Installer (MSI) and InstallScript installations for desktop, server, Web and mobile applications. Installshield allows users to digitally sign installation and application files using a PFX certificate or Windows Key Storage Managed Certificates.

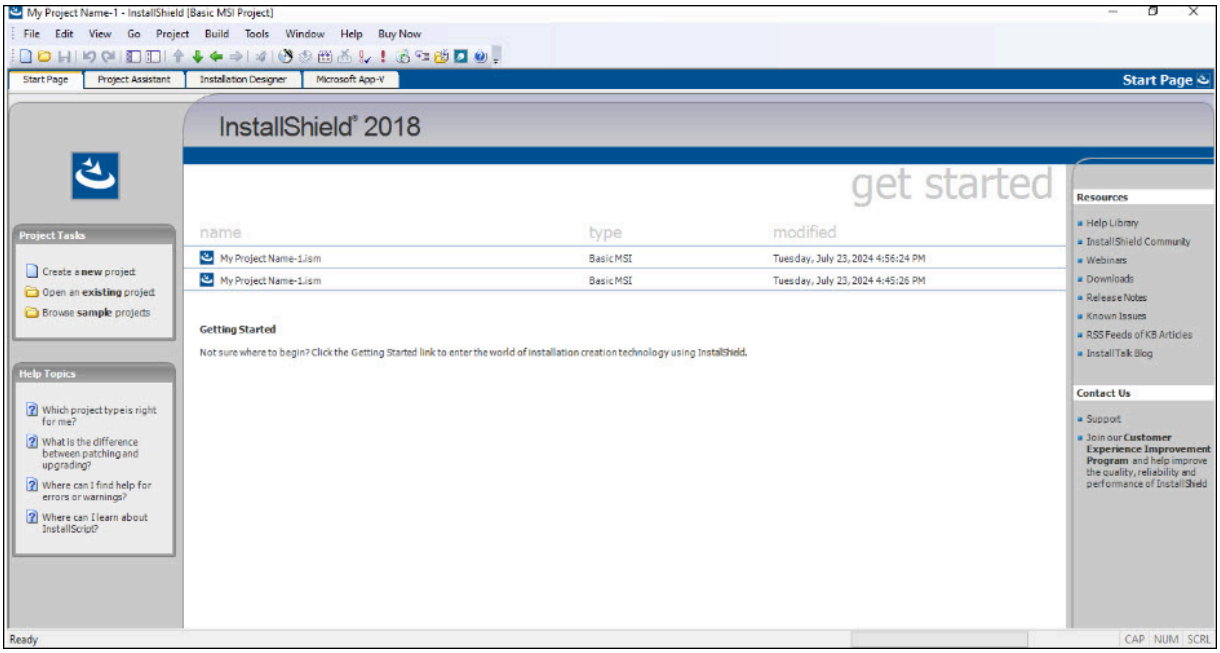
Prerequisites

1. Run the AppViewX SIGN+ Installer to set up the necessary prerequisites for using the AppViewX CSP.
2. Ensure the InstallShield Project is ready for building and signing.

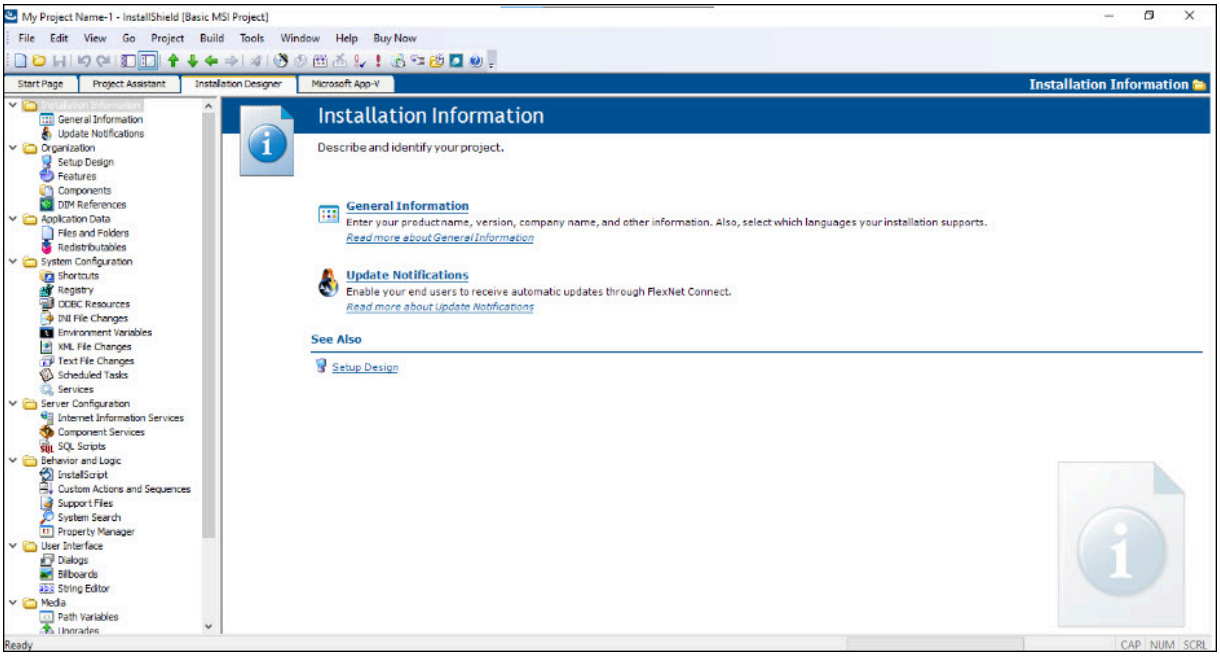
> **Note:** Only Installshield 2015 or above support signing with certificates managed in Windows Key Storage. Versions prior to that support only PFX based signing and hence cannot be integrated to sign using AppViewX SIGN+.

Steps to configure Installshield to sign using AppViewX SIGN+

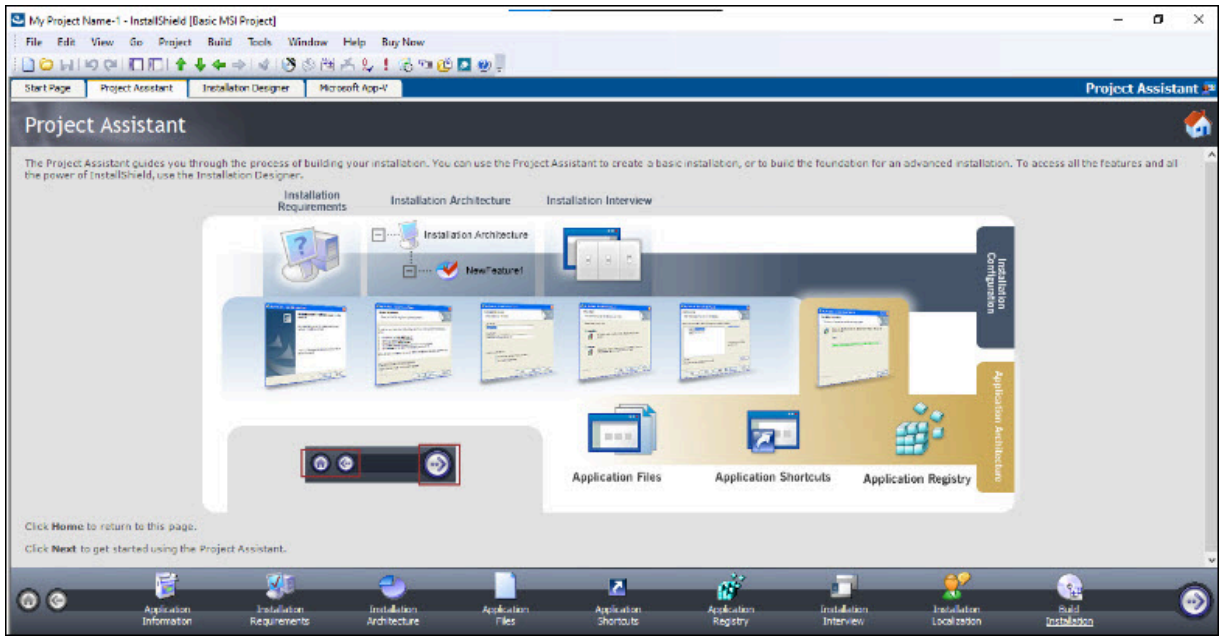
1. Open an existing InstallShield project.



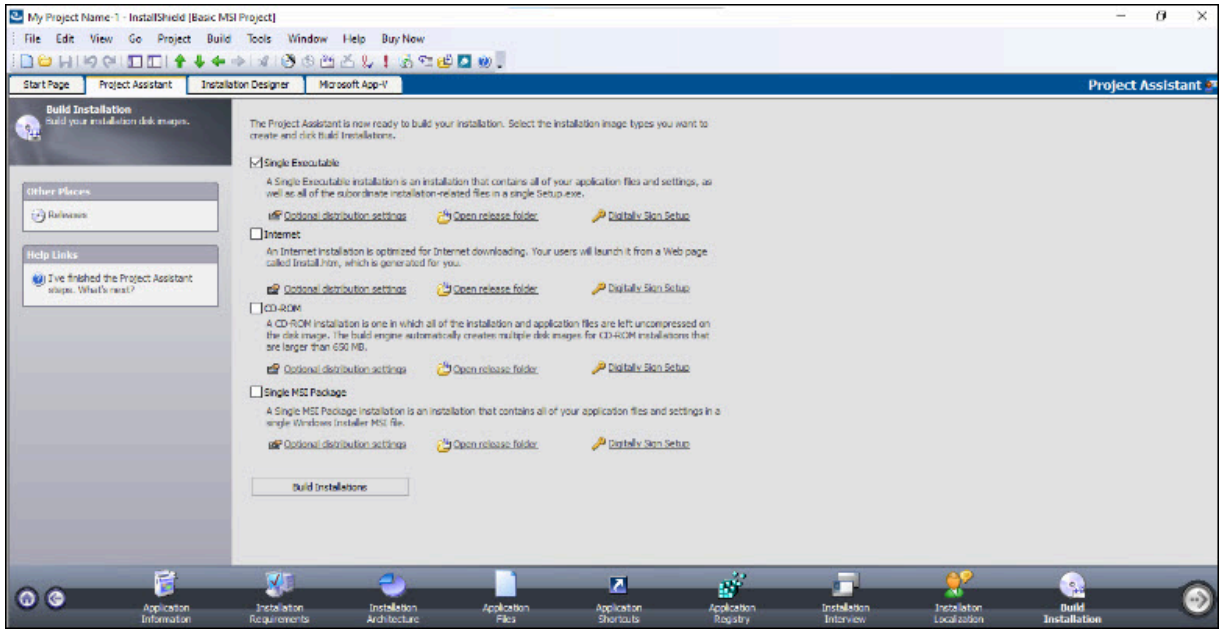
2. View the project in the InstallShield Wizard IDE.



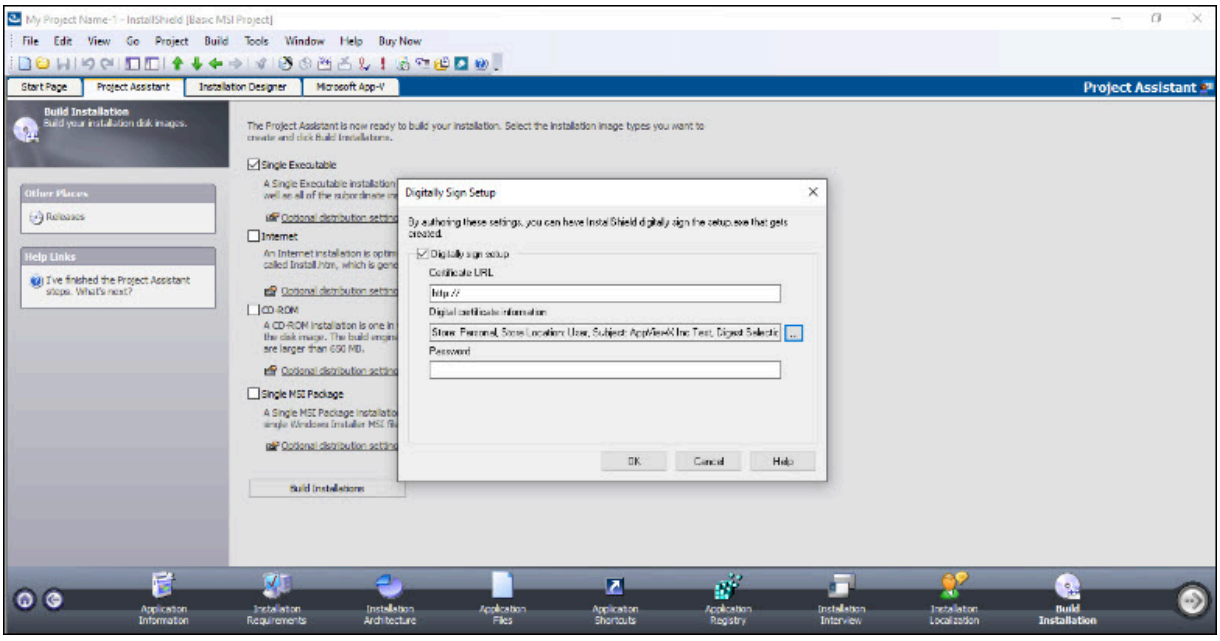
3. Click the **Project Assistant** tab to configure the necessary application settings.



4. Select the **Build Installation** option to configure the building and signing of artifacts.

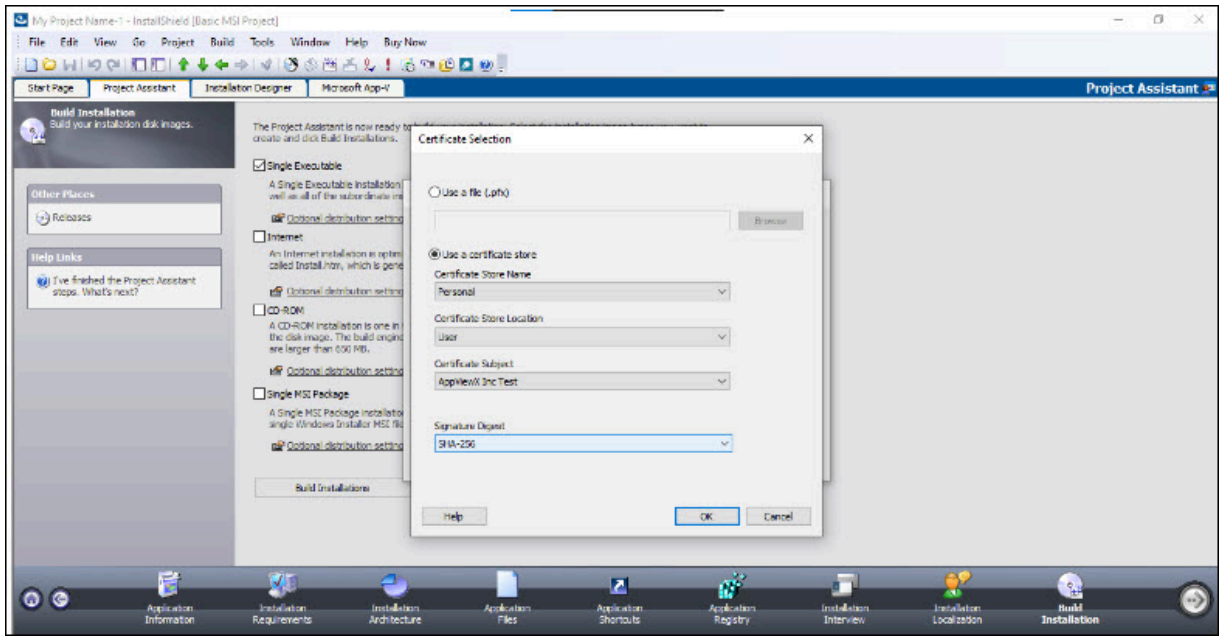


5. For the required output artifact type (e.g., Single Executable in this example), select **Digitally Sign Setup**.

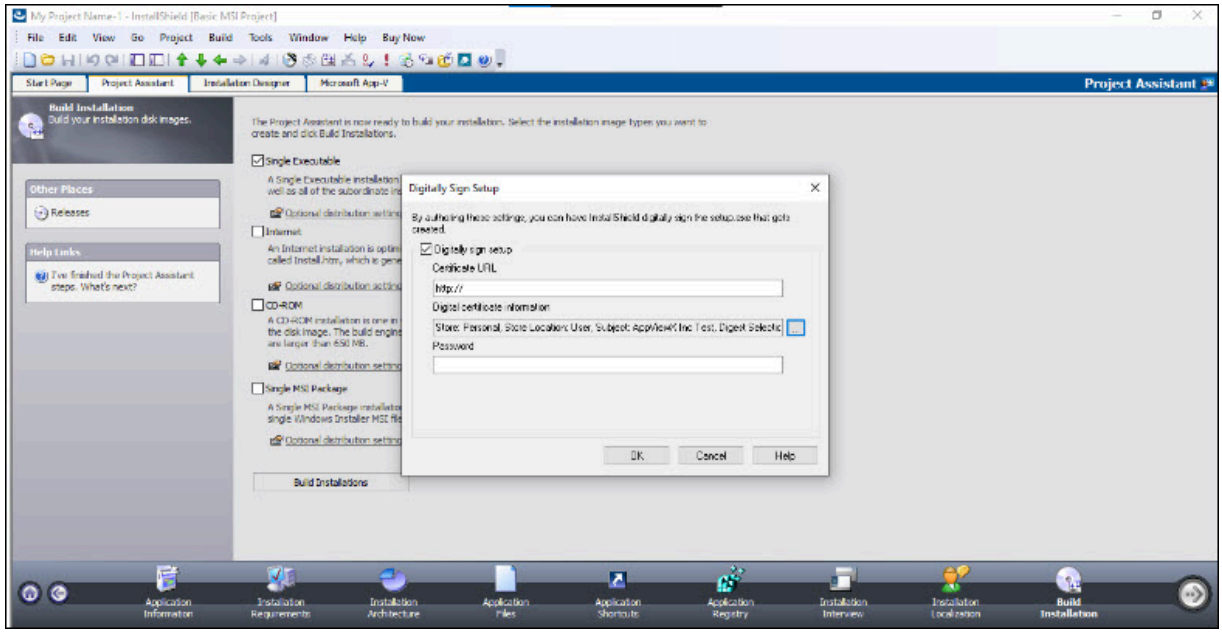


6. Click the option next to **Digital Certificate Information** to select the signing certificate.
7. In the **Certificate Selection** window, select **Use a Certificate Store** and configure the following options:

a.	**Certificate Store Name:**	Personal
b.	**Certificate Store Location:**	User
c.	**Certificate Subject:**	Certificate installed through SIGN+_Installer
d.	**Signature Digest:**	**SHA-256** (Recommended)

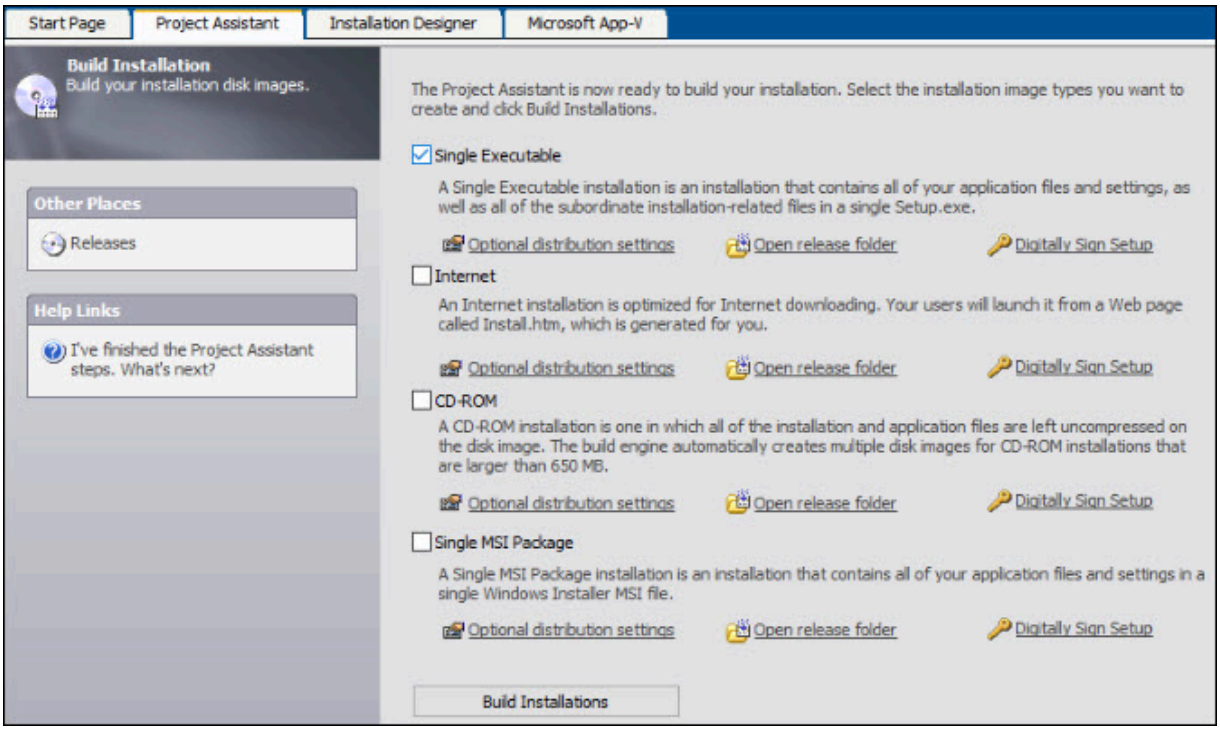


8. Verify the selected options and click **OK**.

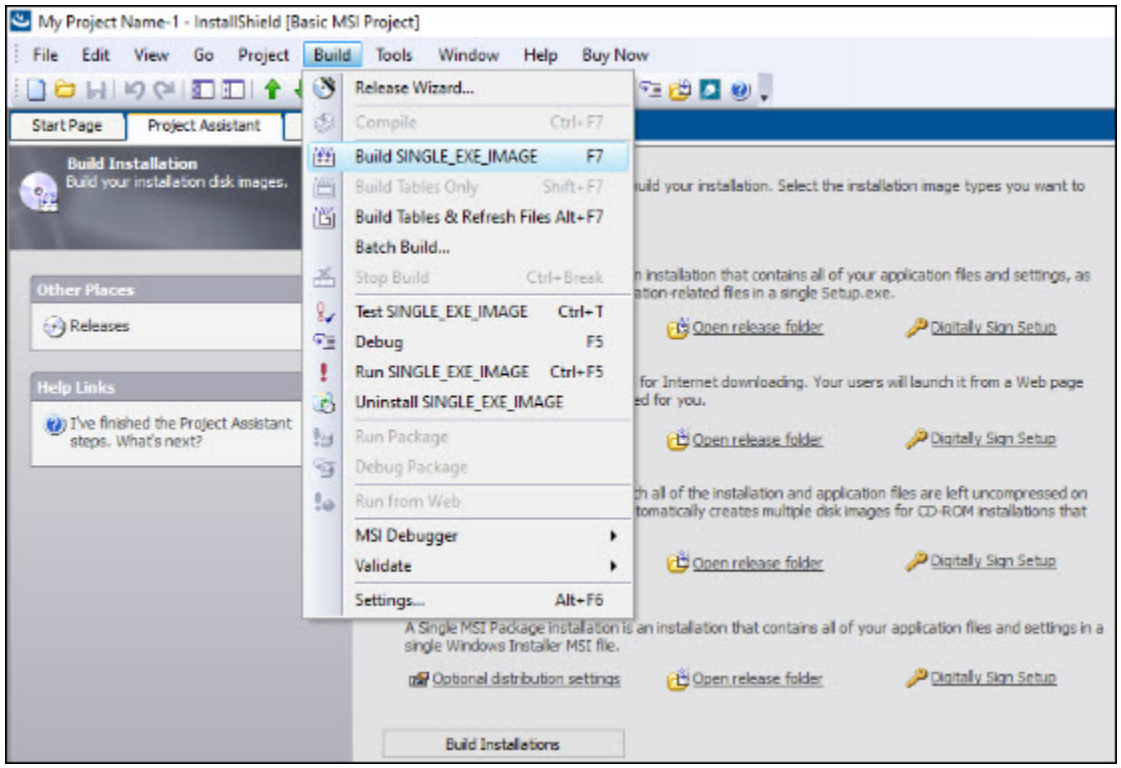


Sign Installer Files: Sample Output

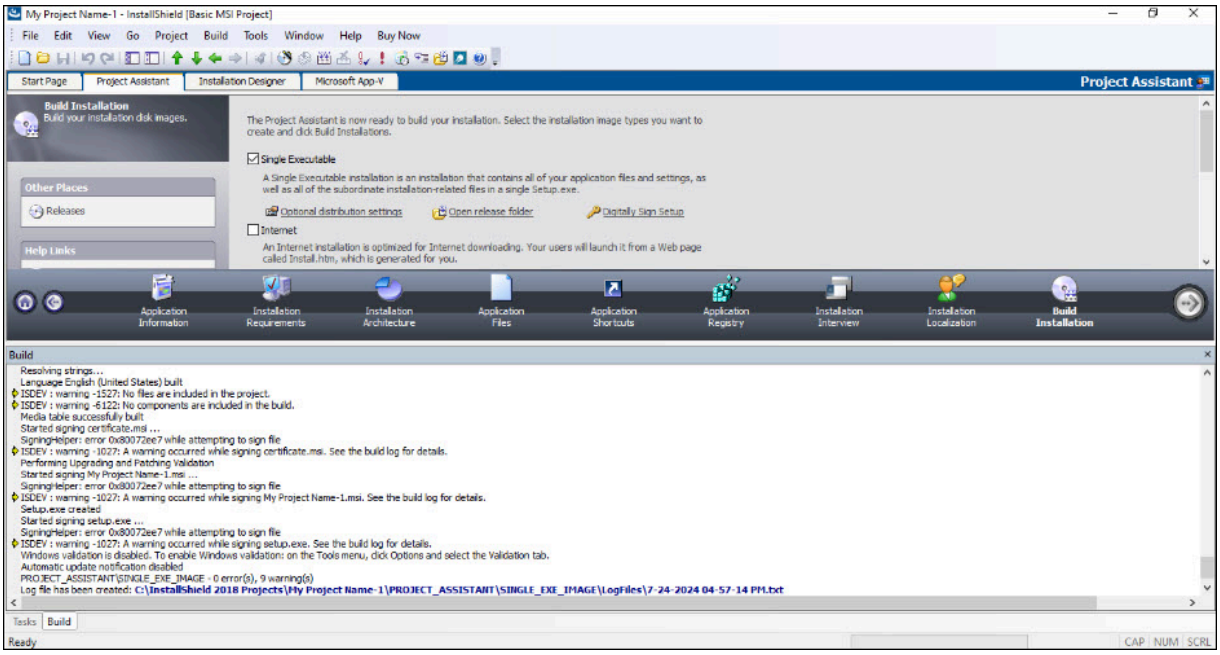
1. Select the desired output type (e.g., Single Executable in this example).



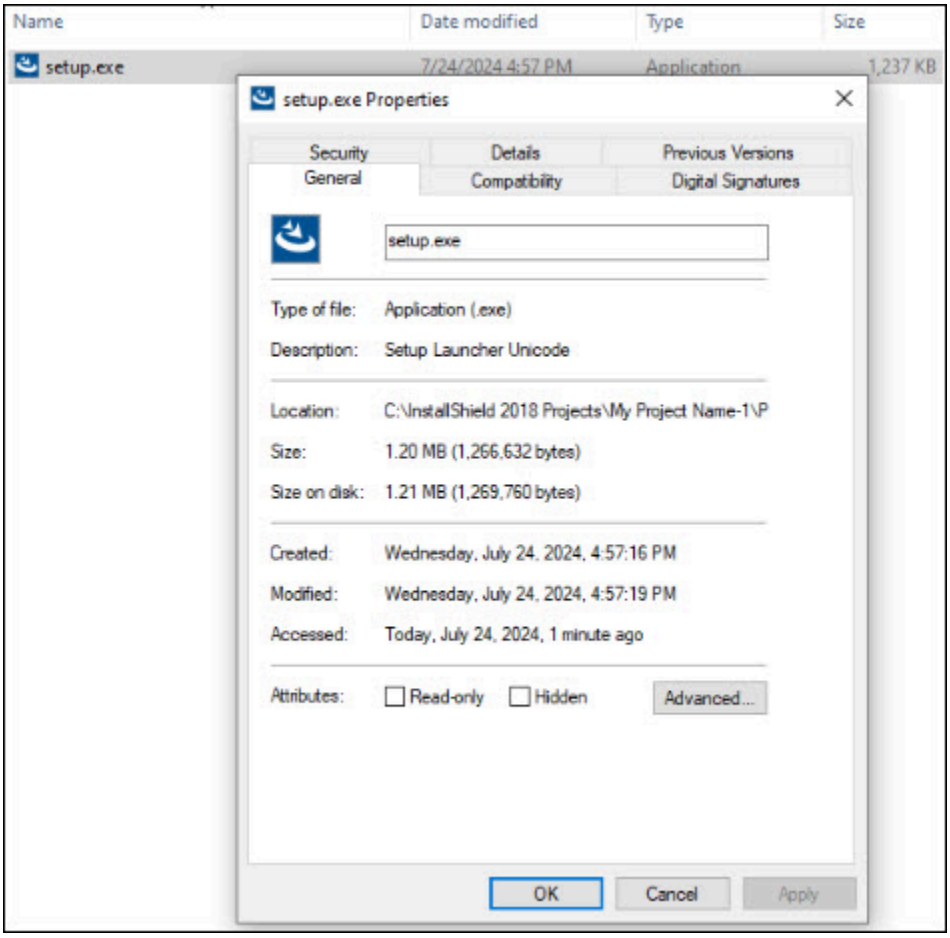
2. Go to **Build -> Build SINGLE_EXE_IMAGE**.



3. Verify the output logs from the build process.



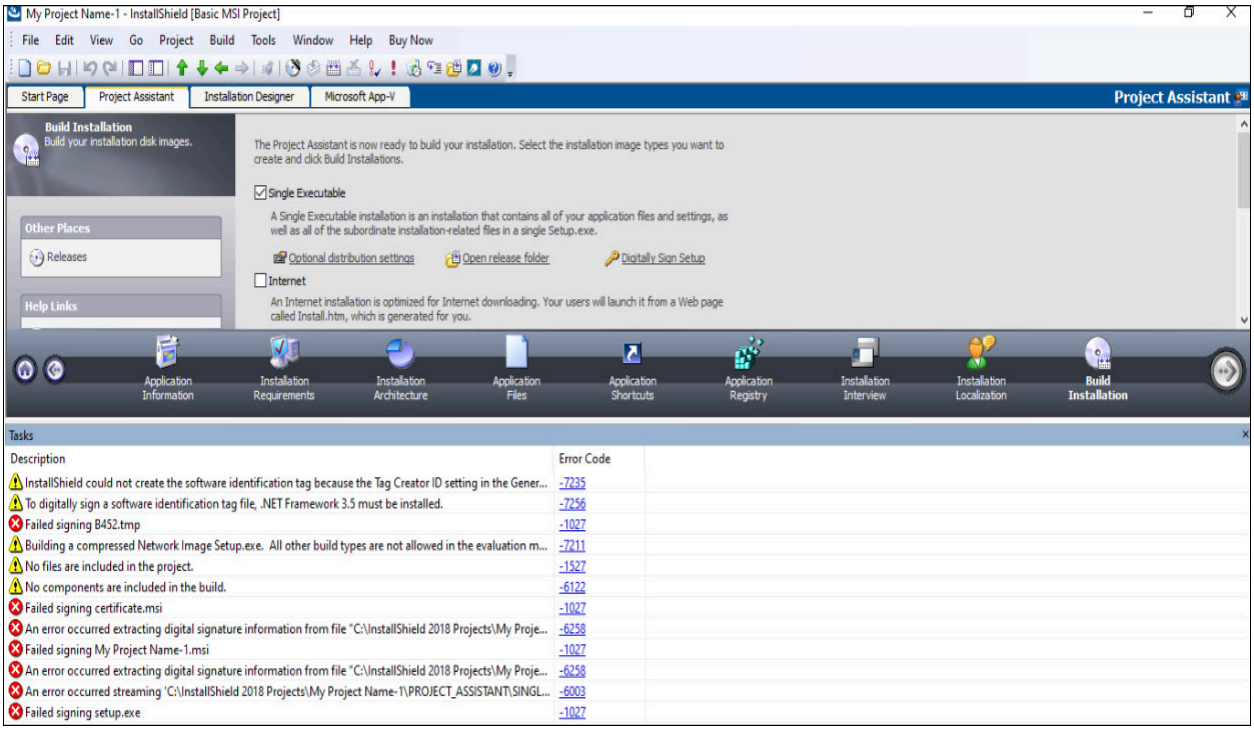
4. Navigate to the Release folder and check the digital signature of the generated file.



Troubleshoot Signing Errors

Error Encountered while signing

Error message:



Problem

This error message occurs due to various reasons like error while establishing connection to the server, authentication error and other validation messages.

Solution

For more information on the error message refer to the AppViewX_CSP_<Day>.log file in **C:\Users \<username>\AppData\Local\Temp** path.

Integrating Code Signing using Scripts

Integrating Code Signing using Scripts provides a step-by-step guide to automating the code signing process through scripts. This resource helps streamline the integration of code signing into your development pipeline, ensuring secure and efficient signing of your software artifacts to meet security and compliance requirements.

AppViewX PKCS#11 Provider Integration with Maven Scripts

Maven is a build automation and project management tool mainly used for Java projects. It uses an XML file `(pom.xml)` to manage project dependencies, build configurations, and project lifecycles, enabling standardized builds and efficient dependency management.

Prerequisites

1. Run the AppViewX SIGN+ Installer to set up the prerequisites for using the AppViewX PKCS#11 Provider with Maven Scripts.
2. Ensure Maven is pre-installed.

Sign

Sample Command Generated in README

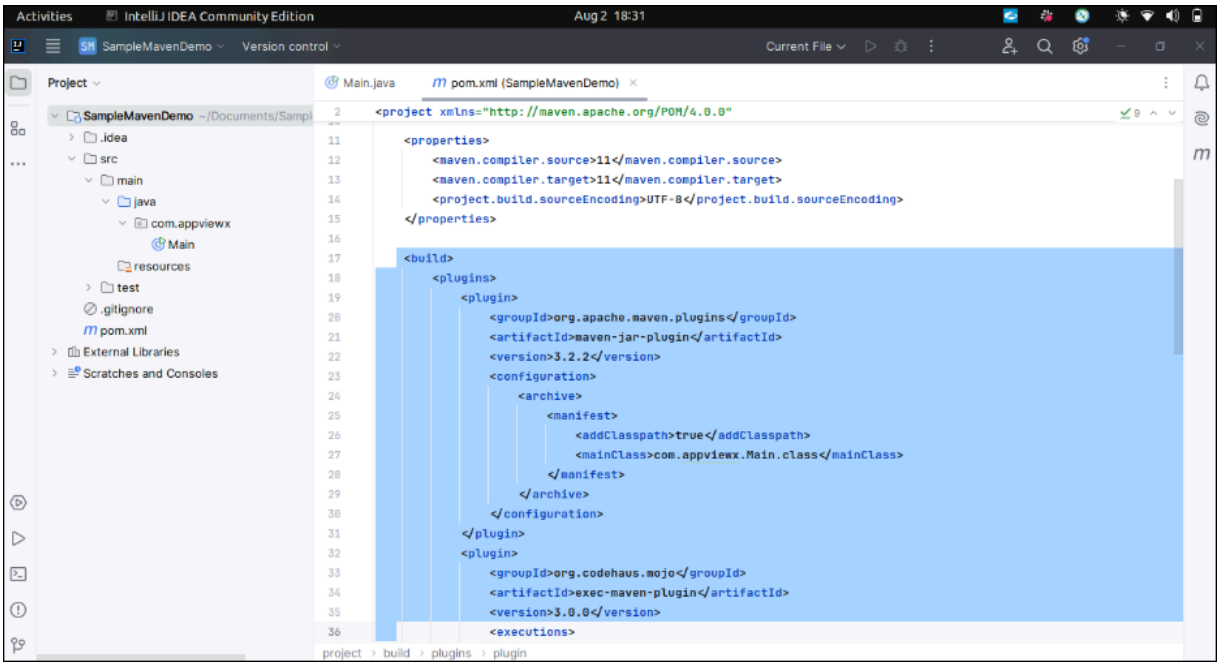
Note: The commands generated by the SIGN+_Installer are not standalone and can only be used in existing Maven/Gradle/Ant Projects for signing.

```
<build>
 <plugins>
  <plugin>
   <groupId>org.apache.maven.plugins</groupId>
   <artifactId>maven-jar-plugin</artifactId>
   <version>3.2.2</version>
   <configuration>
    <archive>
     <manifest>
      <addClasspath>true</addClasspath>
      <mainClass>your.main.class</mainClass>
     </manifest>
    </archive>
   </configuration>
  </plugin>
  <plugin>
   <groupId>org.codehaus.mojo</groupId>
   <artifactId>exec-maven-plugin</artifactId>
   <version>3.0.0</version>
   <executions>
    <execution>
     <id>sign-jar</id>
     <phase>package</phase>
     <goals>
      <goal>exec</goal>
```

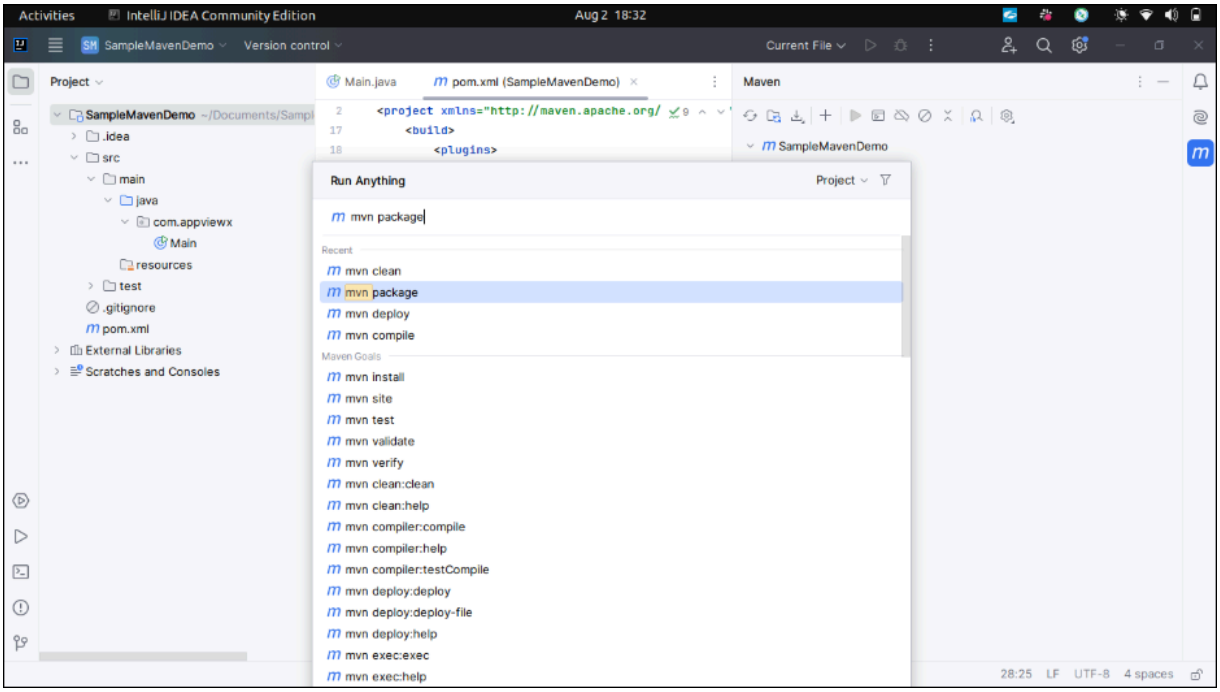
```
        </goals>
        <configuration>
          <executable>jarsigner</executable>
          <workingDirectory>${project.build.directory}</workingDirectory>
          <arguments>
            <argument>-verbose</argument>
            <argument>-keystore</argument>
            <argument>NONE</argument>
            <argument>-storetype</argument>
            <argument>PKCS11</argument>
            <argument>-certs</argument>
            <argument>-providerclass</argument>
            <argument>sun.security.pkcs11.SunPKCS11</argument>
            <argument>-providerArg</argument>
            <argument>/home/admin/AppViewX Sign+/AVXPKCS11V1.cfg</argument>
            <argument>-storepass</argument>
            <argument>12345678</argument>
            <argument>${project.build.finalName}.jar</argument>
            <argument>-signedjar</argument>
            <argument>signedJar.jar</argument>
            <argument>-tsa</argument>
            <argument>http://timestamp.digicert.com</argument>
            <argument>-sigalg</argument>
            <argument>SHA256withRSA</argument>
            <argument>AppViewX Inc Prod's AppViewX Intermediate CA</argument>
          </arguments>
        </configuration>
      </execution>
    </executions>
  </plugin>
 </plugins>
</build>
```

Sample Output

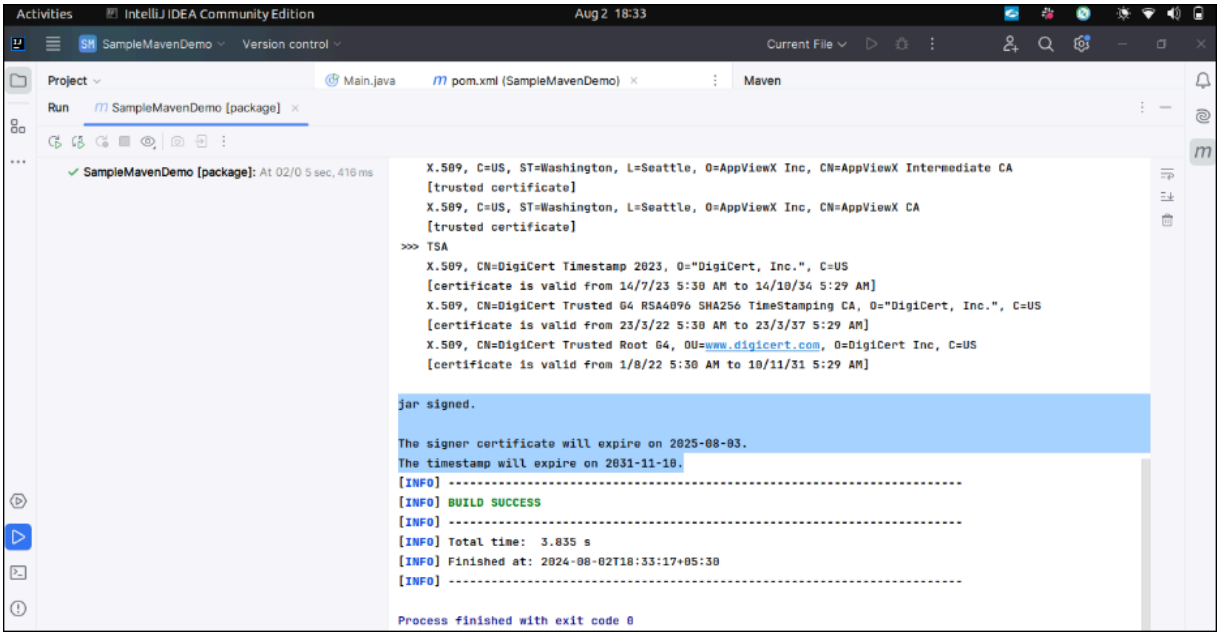
1. Copy the generated README command and paste it into the `pom.xml` file of the relevant project.



2. Execute the `mvn package` command in the command line or through the Maven Command Window in IntelliJ to build and sign the generated JAR file.



3. Check the output window or terminal to verify the status of the build/sign process.



AppViewX PKCS#11 Provider Integration with Gradle Scripts

Gradle is a versatile and robust build automation tool that merges the strengths of Ant and Maven. It utilizes a Groovy-based DSL (Domain-Specific Language) for build configuration and dependency management, supporting incremental builds, advanced dependency handling, and multi-project builds.

Prerequisites

1. Run the AppViewX SIGN+ Installer to set up the prerequisites for using the AppViewX PKCS#11 Provider with Maven Scripts.
2. Ensure Gradle or any supported IDE pre-installed.

Sign

Sample Command Generated in README

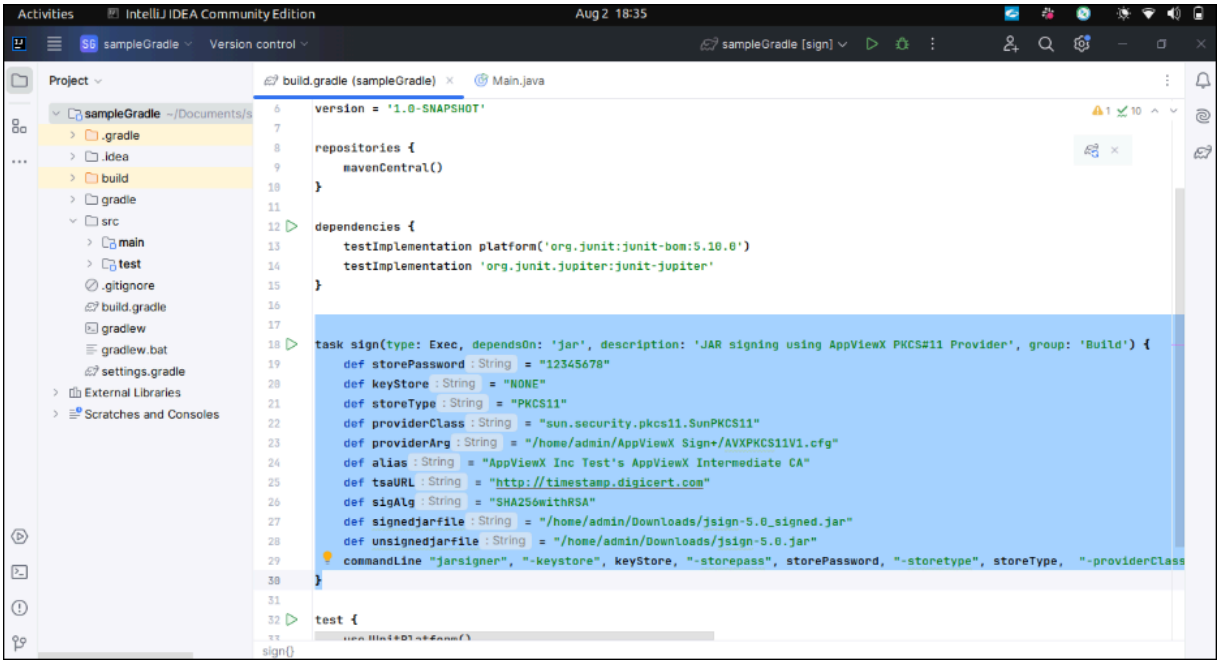
Note: The commands generated by the SIGN+_Installer are not standalone and can only be used in existing Maven/Gradle/Ant Projects for signing.

```
task sign(type: Exec, dependsOn: 'jar', description: 'JAR signing using AppViewX PKCS#11 Provider', group: 'Build') {

  def storePassword = "12345678"

  def keyStore = "NONE"
```

```
  def storeType = "PKCS11"

  def providerClass = "sun.security.pkcs11.SunPKCS11"

  def providerArg = "/home/admin/AppViewX Sign+/AVXPKCS11V1.cfg"

  def alias = "AppViewX Inc Prod's AppViewX Intermediate CA"

  def tsaURL = "http://timestamp.digicert.com"

  def sigAlg = "SHA256withRSA"

  def signedjarfile = "<output_file_path>"

  def unsignedjarfile = "<input_file_path>"

  commandLine "jarsigner", "-keystore", keyStore, "-storepass", storePassword, "-storetype", storeType,  "-providerClass", providerClass, "-providerArg",
providerArg, unsignedjarfile, "-signedjar", signedjarfile, "-tsa", tsaURL, "-sigalg", sigAlg, alias
}
```

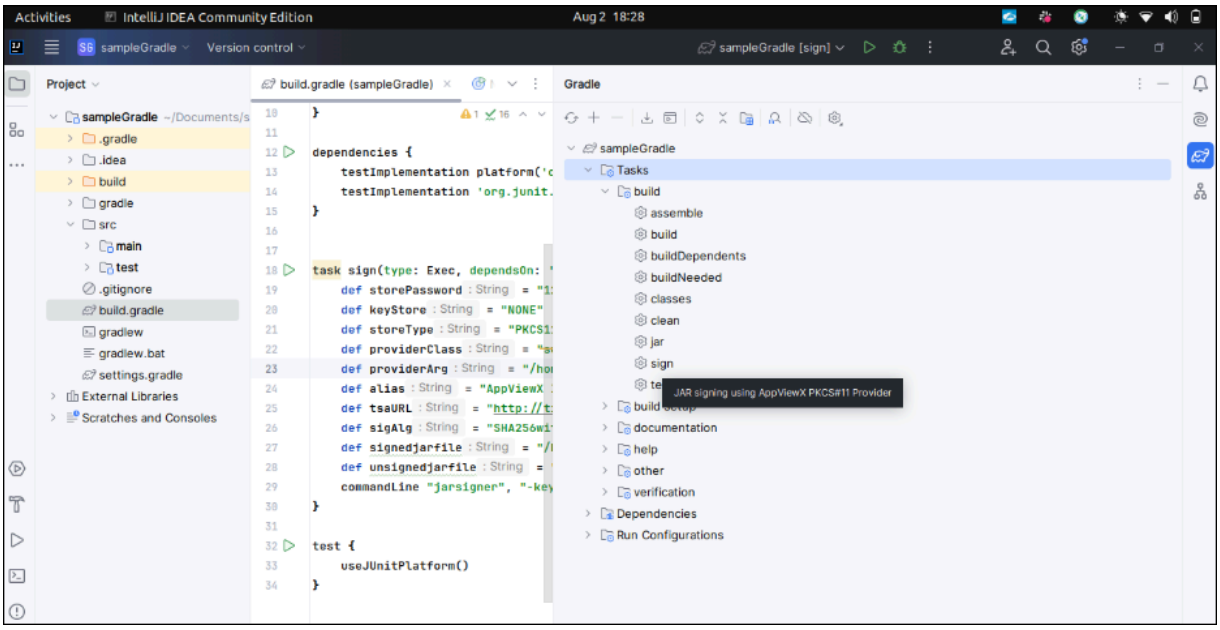
Sample Output

1. Copy the generated README command and paste it into the `build.gradle` file of the relevant project.

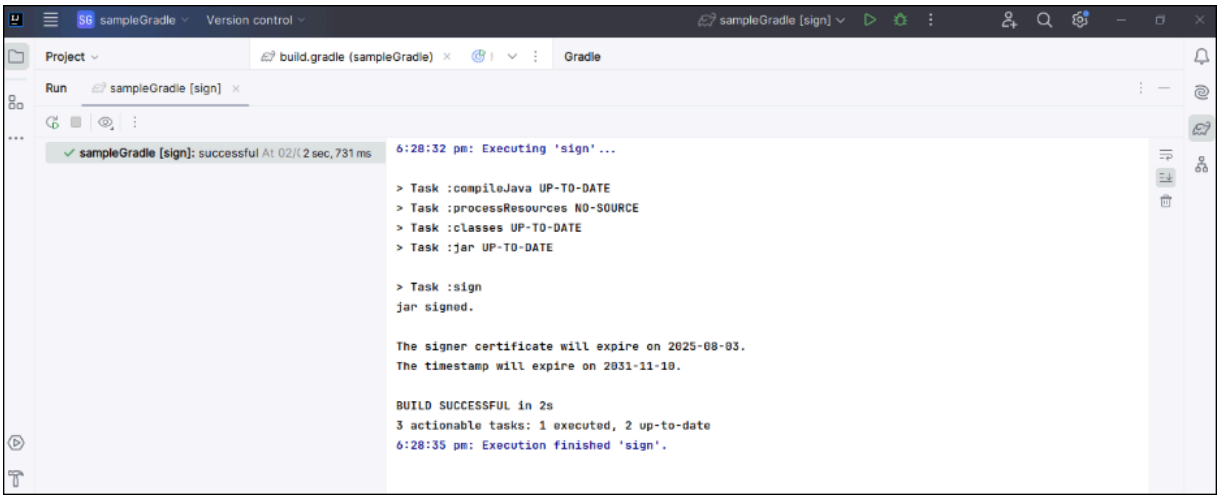


> **Note:** Replace the **<input_file_path>** and **<output_file_path>** in the `build.gradle` file.

2. Run the `sign` task from the Gradle menu to execute the script.



3. Check the output window or terminal to verify the status of the build/sign process.



AppViewX PKCS#11 Provider Integration with Ant Scripts

Ant is a Java-based build tool that utilizes XML configuration files (build.xml) to define build processes. It is more script-oriented and less prescriptive than Maven and Gradle, offering a flexible approach to automating build tasks without enforcing a specific project structure or dependency management system.

Prerequisites

1. Run the AppViewX SIGN+ Installer to set up the prerequisites for using the AppViewX PKCS#11 Provider with Maven Scripts.
2. Ensure Ant and Eclipse IDE are pre-installed.

Sign

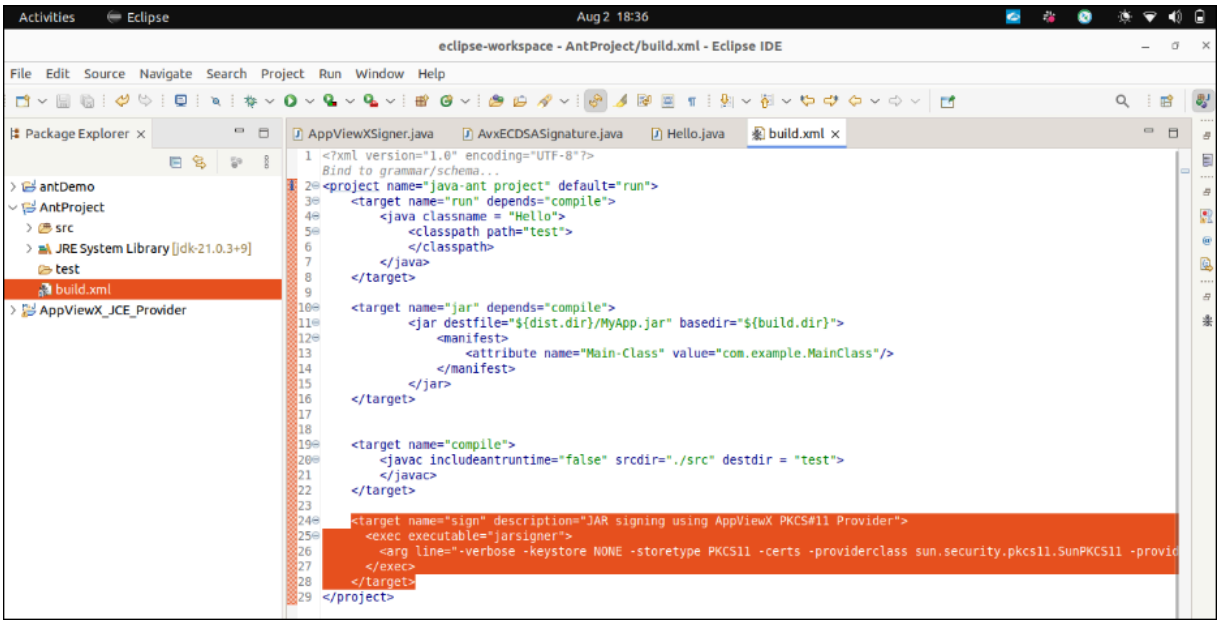
Sample Command Generated in README

Note: The commands generated by the SIGN+_Installer are not standalone and can only be used in existing Maven/Gradle/Ant Projects for signing.

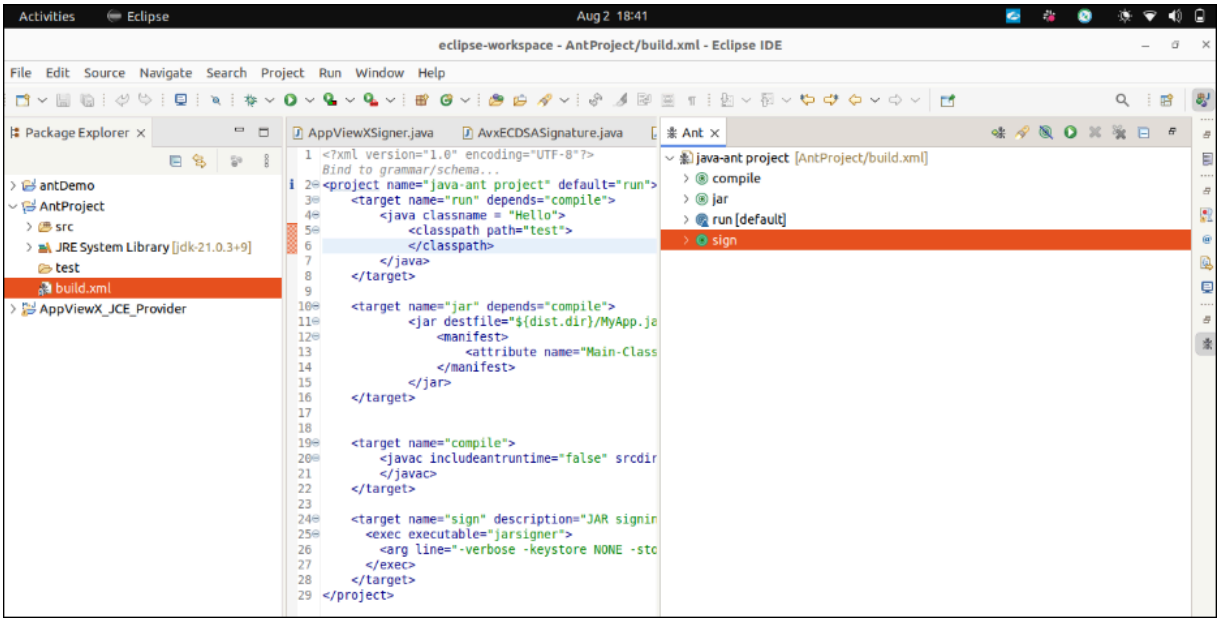
```
<target name="sign" description="JAR signing using AppViewX PKCS#11 Provider">
 <exec executable="jarsigner">
  <arg line="-verbose -keystore NONE -storetype PKCS11 -certs -providerclass sun.security.pkcs11.SunPKCS11 -providerArg &amp;quot;/home/admin/AppViewX Sign+/AVXPKCS11V1.cfg&amp;quot; -storepass 12345678 input_file_path -signedjar output_file_path -tsa &amp;quot;http://timestamp.digicert.com&amp;quot; -sigalg &amp;quot;SHA256withRSA&amp;quot; &amp;quot;AppViewX Inc Prod's AppViewX Intermediate CA&amp;quot;" />
 </exec>
</target>
```

Sample Output

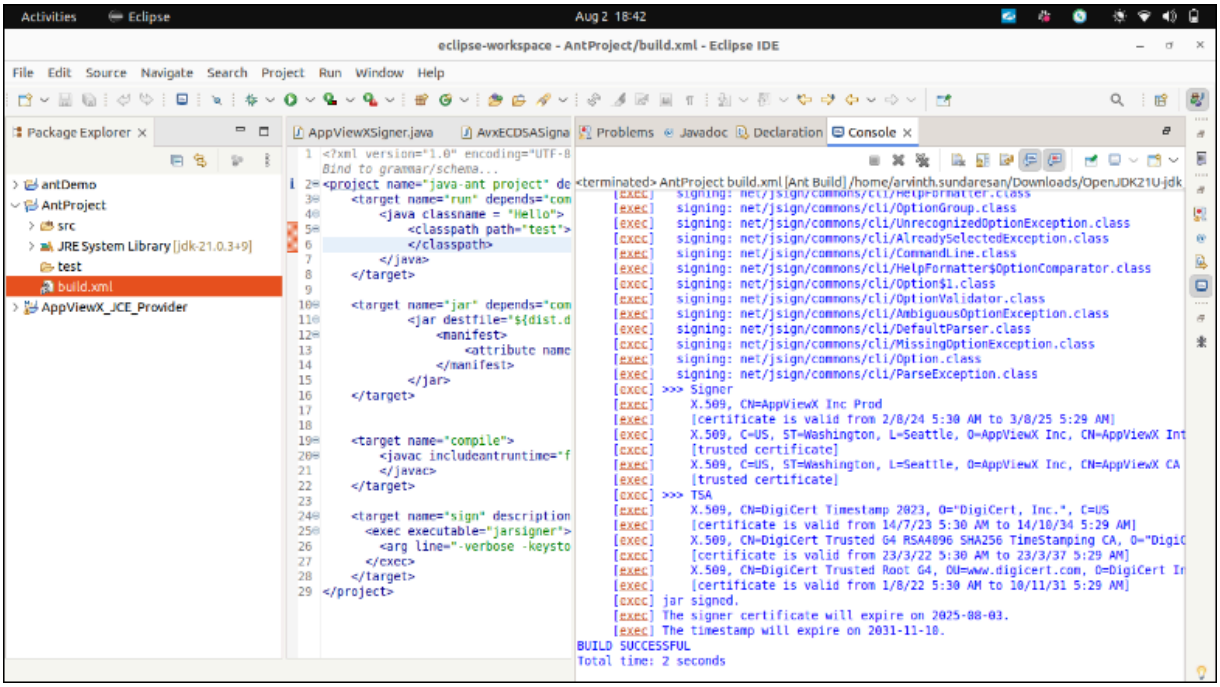
1. Copy the generated README command and paste it into the `build.xml` file of the relevant project.
2. Modify the `input_file_path` and `output_file_path` according to your requirements.



3. Run the `sign` task from the Ant window to build and sign the file.



4. Check the output window or terminal to verify the status of the build/sign process.



ClickOnce Deployment with Visual Studio

ClickOnce is a deployment technology that allows users to create self-updating Windows applications with minimal user interaction. Visual Studio fully supports publishing and updating ClickOnce-deployed applications for projects developed in Visual Basic and Visual C#. It also integrates natively to sign artifacts using certificates managed in Windows Key Storage.

Download Visual Studio

1. Download from [Download Visual Studio Tools - Install Free for Windows, Mac, Linux](#) for Windows.
2. Install Visual Studio using the Installation Wizard.

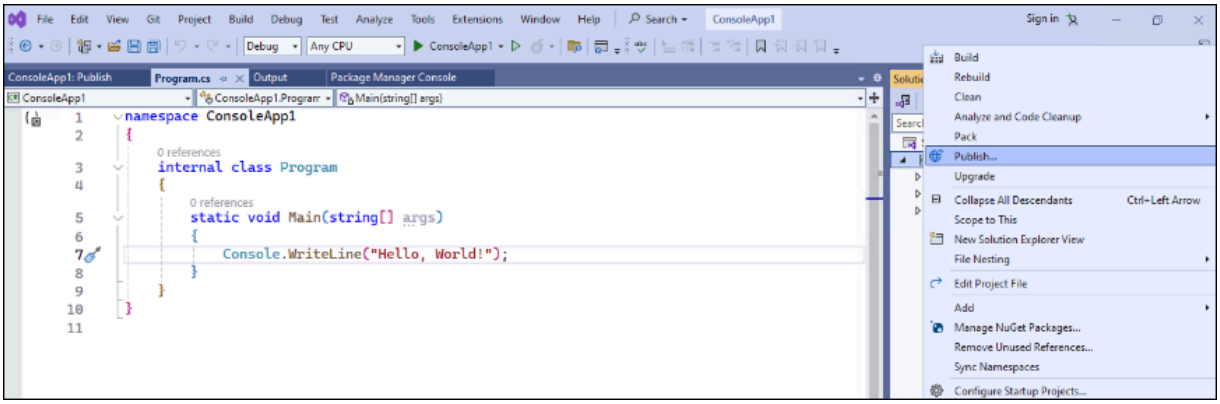
Sign ClickOnce Manifest Files in Microsoft Visual Studio using AppViewX CSP

Prerequisites:

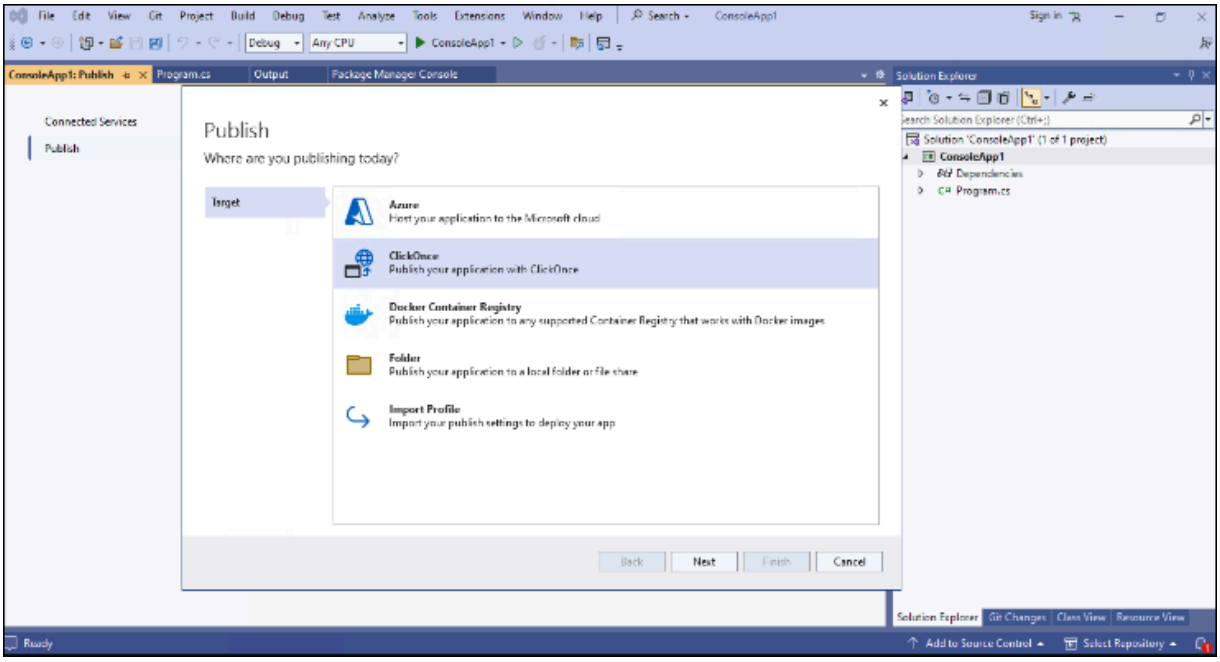
1. Run the AppViewX SIGN+ Installer to set up the prerequisites for using the AppViewX CSP.
2. Ensure Visual Studio is installed and that you have a ClickOnce-supported project ready for publishing.

Sign ClickOnce Manifest files:

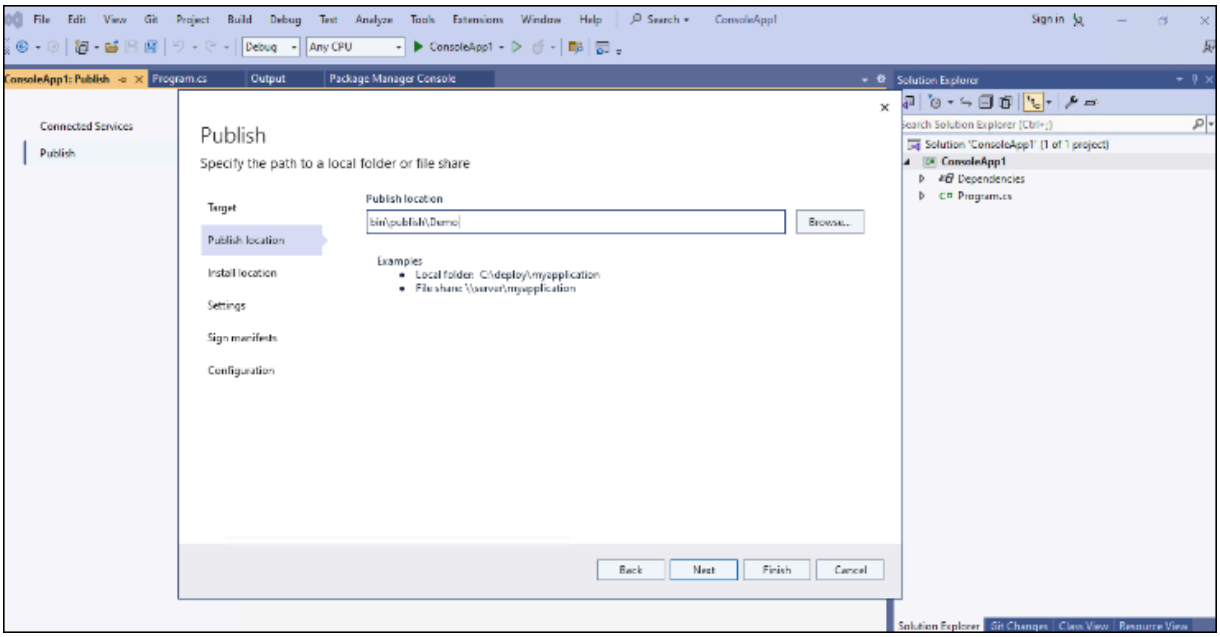
1. Open the project in Visual Studio that supports ClickOnce deployment.



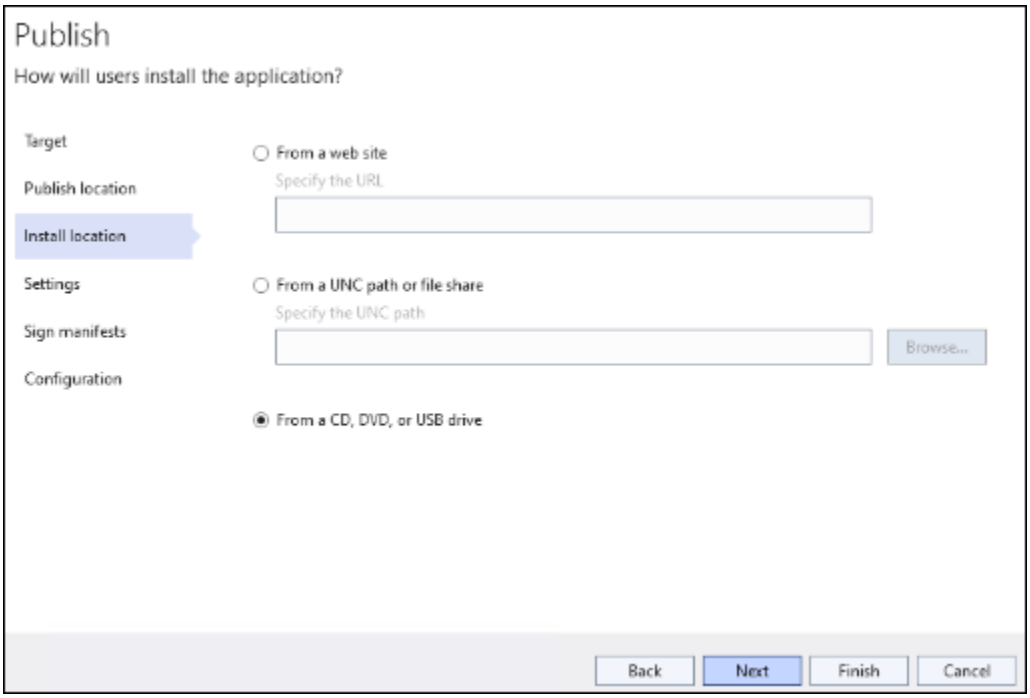
2. In the Solution Explorer tab, right-click the project and select "Publish" to create a Publish Profile for the project.
3. Select the ClickOnce option.



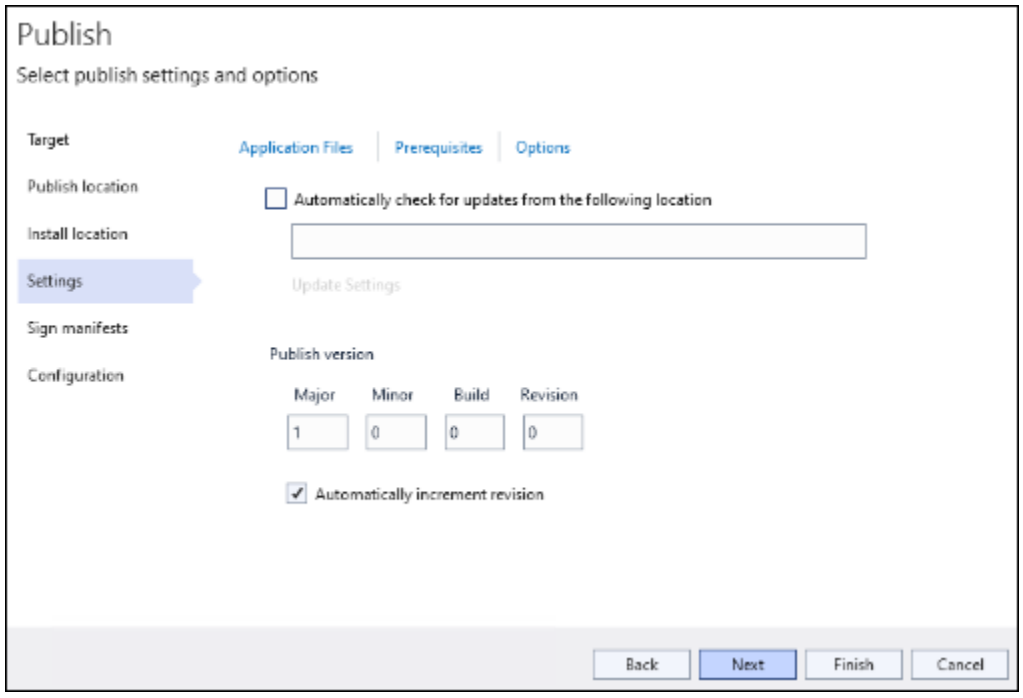
4. In the **Publish location** section, enter the output location for the artifacts to be generated.



5. In the **Install location** section, select the **From CD, DVD or USB Drive** option.

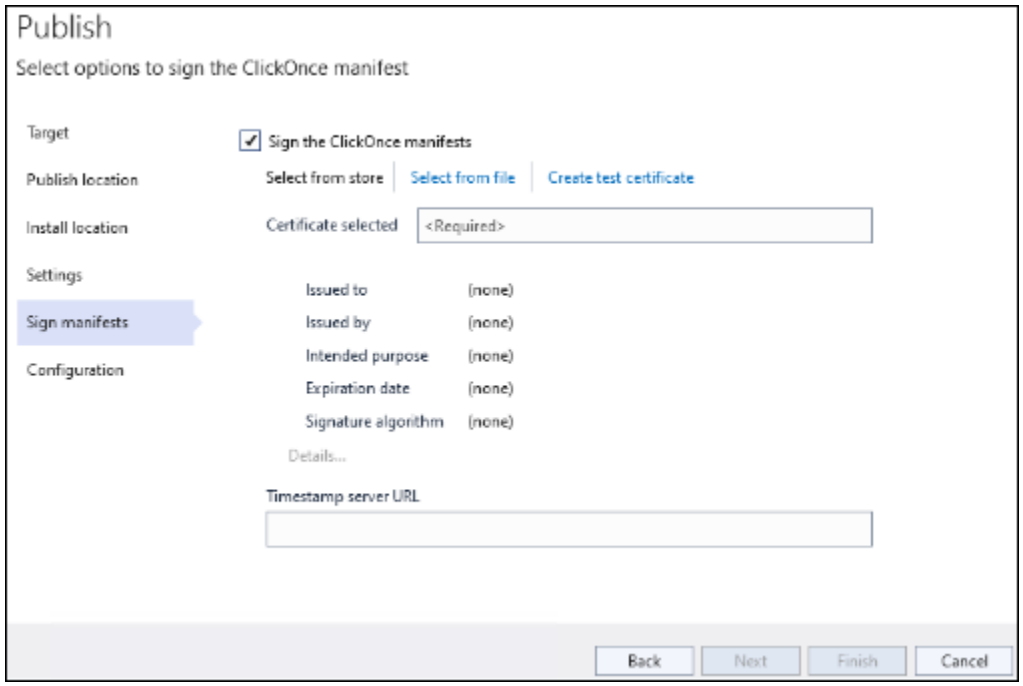


6. In the **Settings** section, specify the required publish settings for the profile.

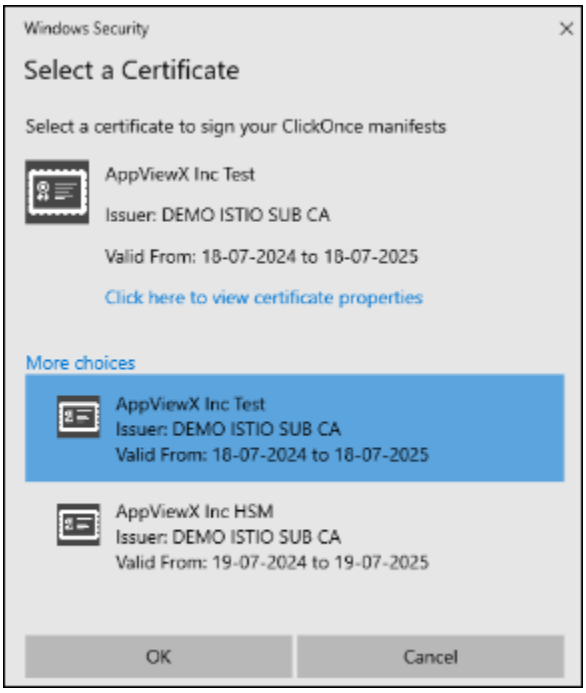


7. In the **Sign Manifests** section, select the **Sign the ClickOnce manifest** checkbox and select the **Select from Store** option to sign with a certificate from the Windows Key Storage.

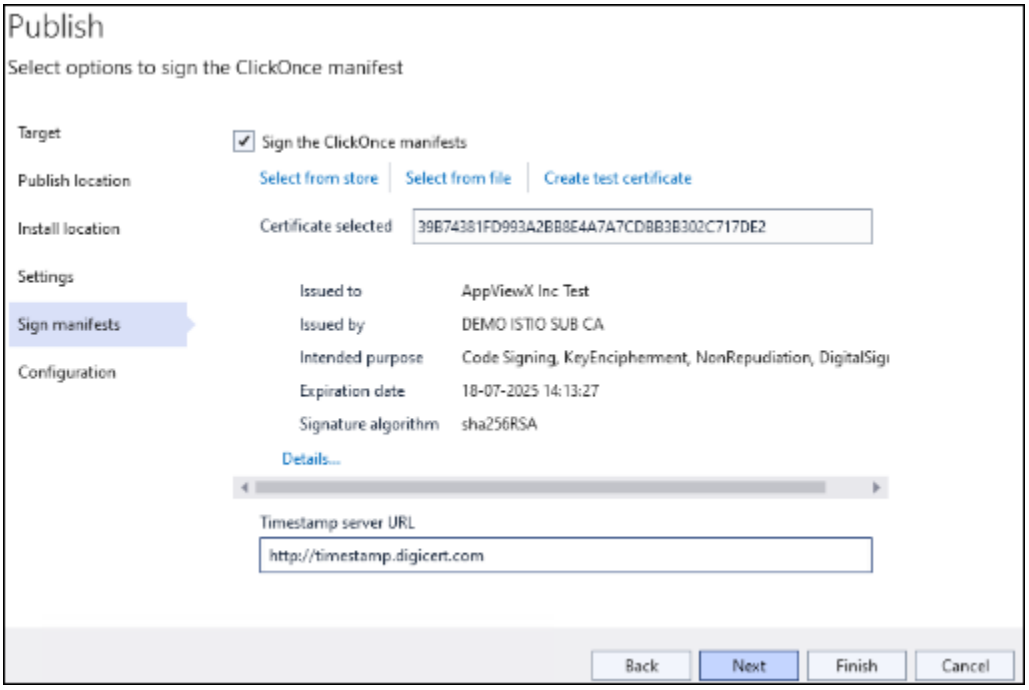
Note: ClickOnce internally uses Mage and hence supports only RSA Keys.



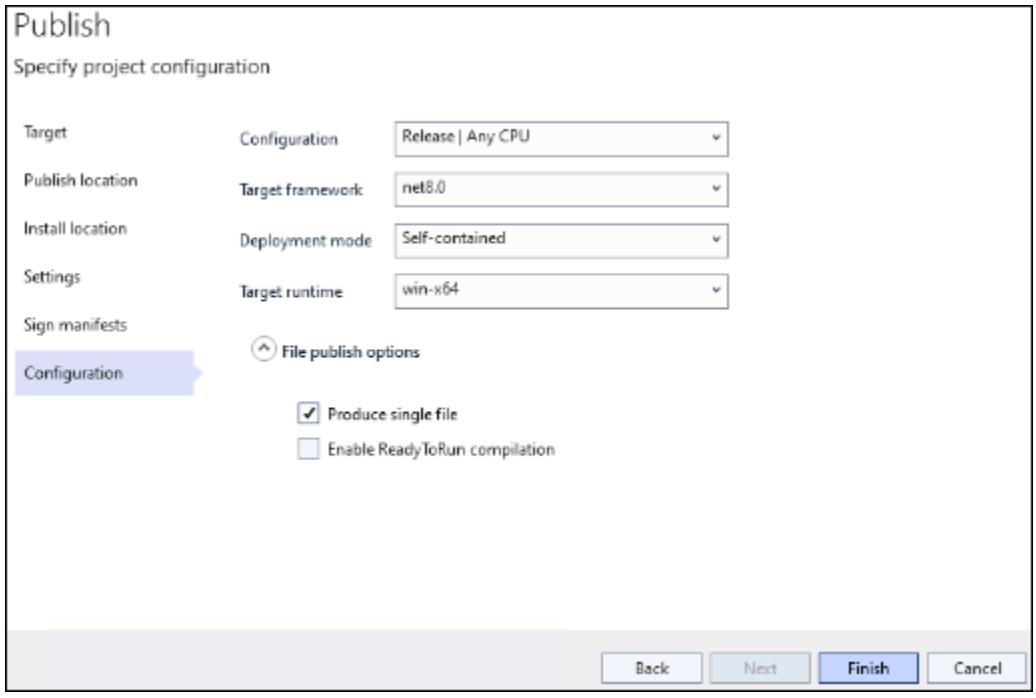
8. In the Windows Security popup, select the same signing certificate (which you selected when downloading the SIGN+ package) and click **OK**.



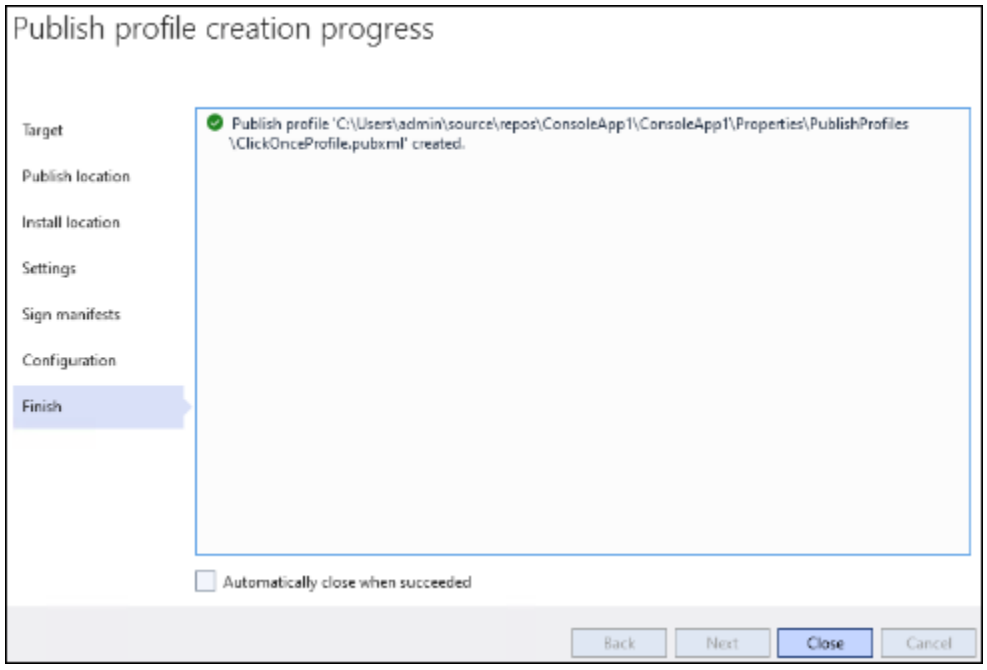
9. Enter the Timestamp Server URL. The URL configured in the Signing Policy can be found in the README generated after running the SIGN+ Installer.



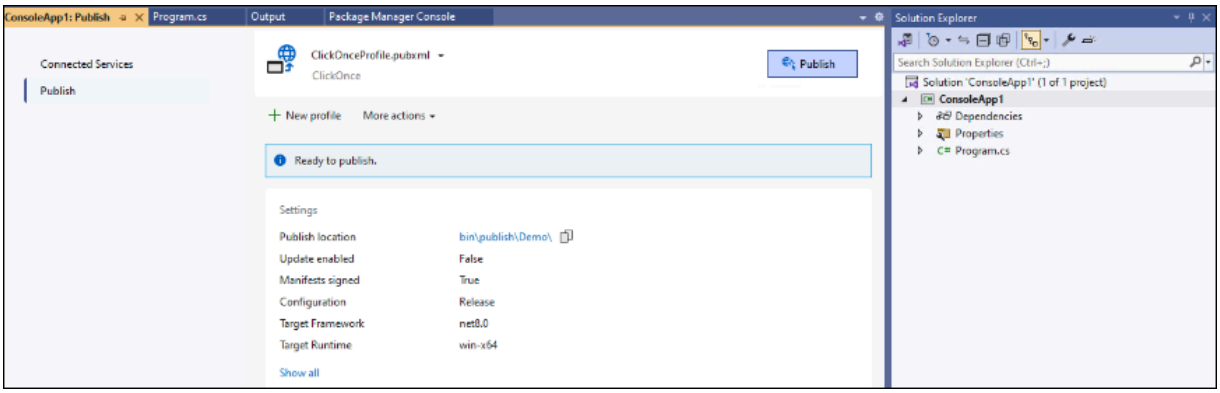
10. In the **Configuration** section, select the required File Publish options.



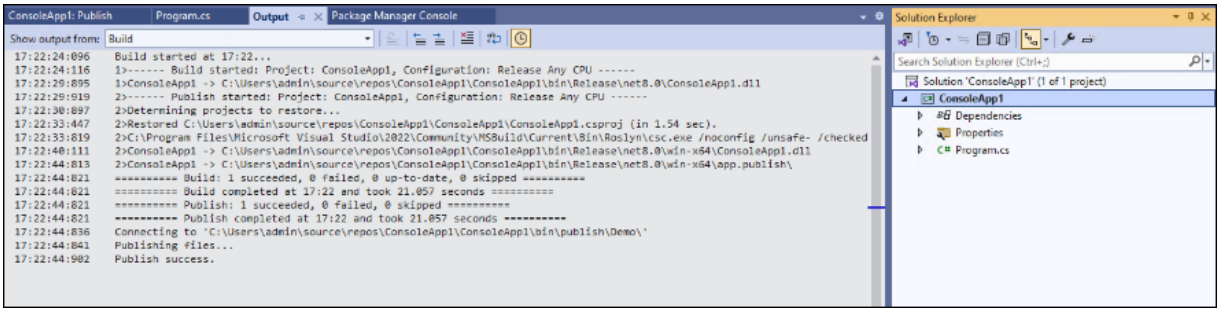
11. Click **Finish** and then **Close** to create the publish profile.



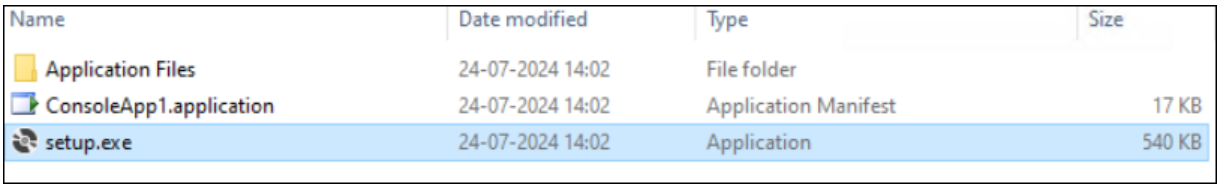
12. To sign the deployment files using ClickOnce, select the created publish profile and click **Publish**.



13. View the Build and Publish status in the output logs.



14. Navigate to the publish directory and verify the digital signature of the generated artifacts.



Note: By default only SHA256 is used for hashing.

Note: When signing using ClickOnce Deployment Mode, multiple signature units will be consumed depending on the number of file references within the project.

For the signing of manifest files, two signature units are consumed in AppViewX SIGN+. This is because two hash signatures are created for different types of content within the manifest:

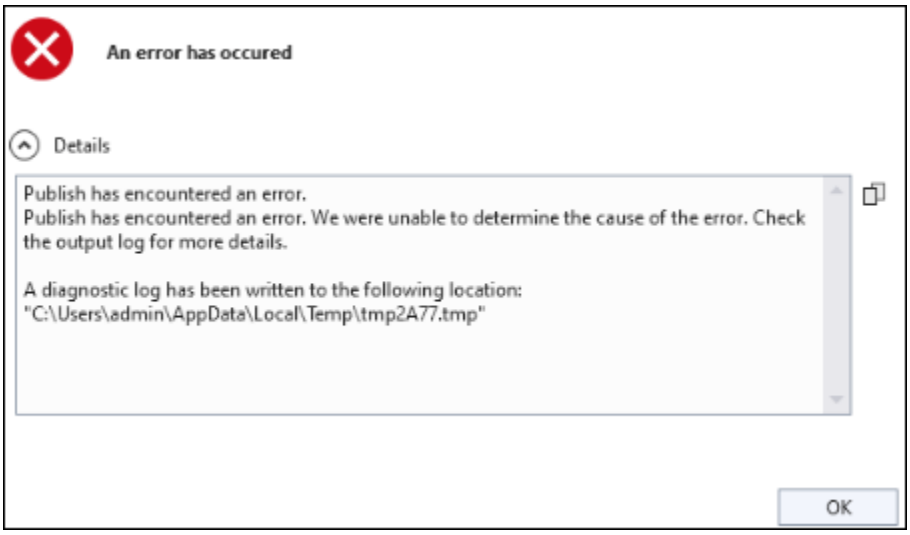
- The first hash signature is generated for the manifest file itself.
- The second hash signature is generated for the files referenced within the manifest.

This double-signing process ensures the integrity of both the manifest file and its referenced files, safeguarding against any tampering attempts.

Troubleshoot Signing Errors

Error Encountered while signing.

Error message:



Problem:

This error message occurs due to various reasons like error while establishing connection to the server, authentication error and other validation messages.

Solution:

For more information on the error message refer to the AppViewX_CSP_<Day>.log file in **C:\Users\<username>\AppData\Local\Temp** path.

Integrating SIGN+ for Document Signing

AppViewX SIGN+ currently ensures the authenticity and integrity of code artifacts, such as executables and script files, through cryptographic signatures. To address the growing need for secure and verifiable document handling, SIGN+ is enhanced to support document signing for PDFs and Microsoft Office files. Integrating SIGN+ for document signing streamlines your workflow by providing a seamless solution for secure and efficient signature collection. This will utilize the existing AppViewX Cryptographic Service Provider (CSP) library to apply digital signatures to these document types. By incorporating document signing, SIGN+ will provide a unified platform that meets user's evolving security needs, enabling both code and documents to be reliably trusted and verified.

Integrating Adobe Acrobat

Adobe Acrobat is a software application developed by Adobe Inc. that allows users to create, view, manipulate, print, and manage files in Portable Document Format (PDF). It is widely used for sharing documents because PDFs retain their formatting across different devices and platforms. Adobe Acrobat offers various tools for editing text and images, converting documents to and from PDF, and ensuring document security.

Download Adobe Acrobat

1. Download from Download Adobe Acrobat Reader: Free PDF viewer for Windows.
2. Install using the Installation Wizard.

Sign PDF Files with Adobe Acrobat using AppViewX CSP

Signing PDF files is essential for validating the authenticity and integrity of documents in digital workflows. It ensures that the content has not been altered and confirms the identity of the signer, which is crucial for legal, financial, and business transactions. Adobe Acrobat facilitates this process by providing robust tools for adding digital signatures. With Adobe Acrobat, users can easily insert their signatures, whether by typing, drawing, or uploading an image, ensuring a seamless and secure signing experience. Additionally, Adobe Acrobat supports advanced digital signature features, such as certificate-based signatures, which offer enhanced security and compliance with legal and regulatory standards.

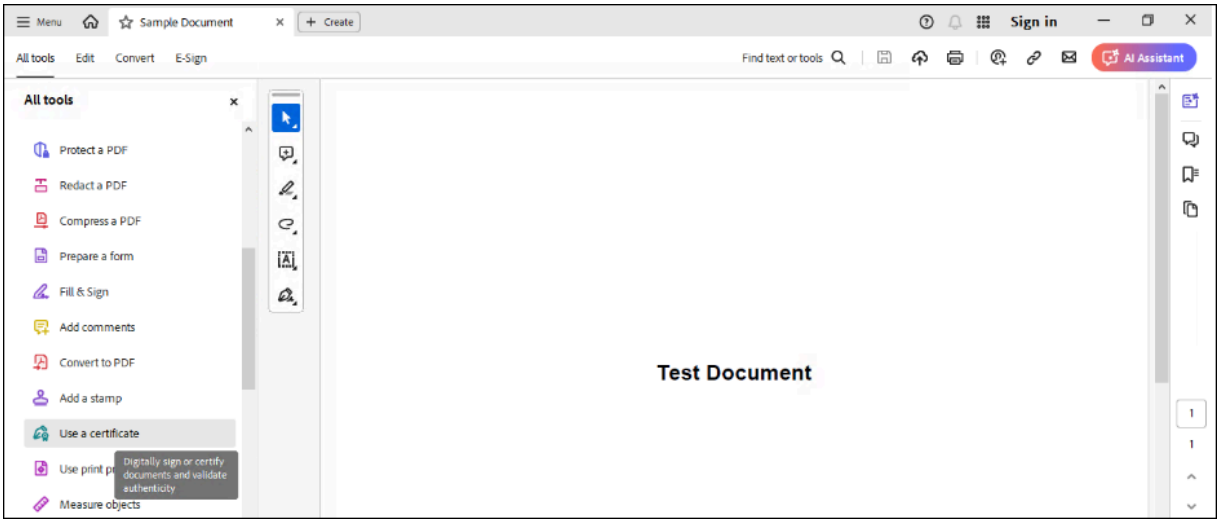
For more information on Certificate based signatures using Adobe Acrobat, refer Certificate-based signatures, Adobe Acrobat.

Prerequisites:

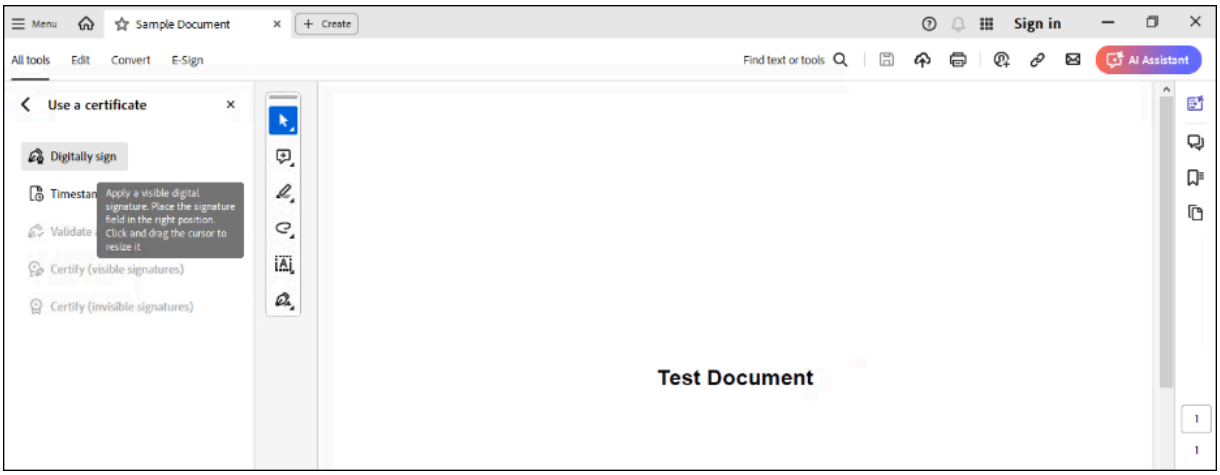
1. Run the AppViewX SIGN+ Installer to set up the prerequisites for using the AppViewX CSP.
2. Ensure Adobe Acrobat Reader is Installed.

Add Certificate-based Signature to a PDF

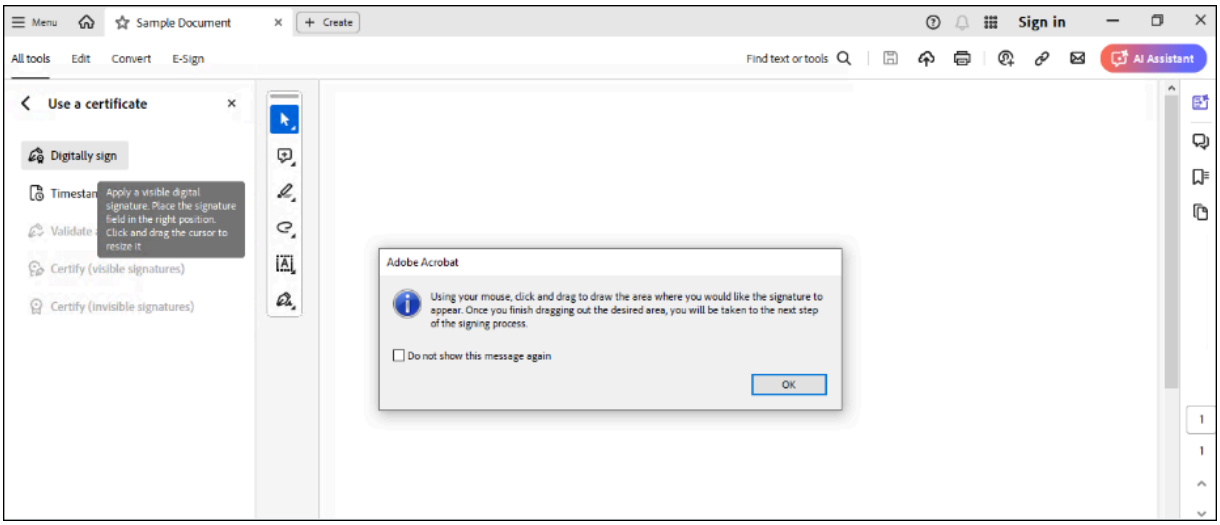
1. Open the PDF to be signed in Adobe Acrobat and go to **All Tools** -> **View More** -> **Use a Certificate** in the Global Bar.



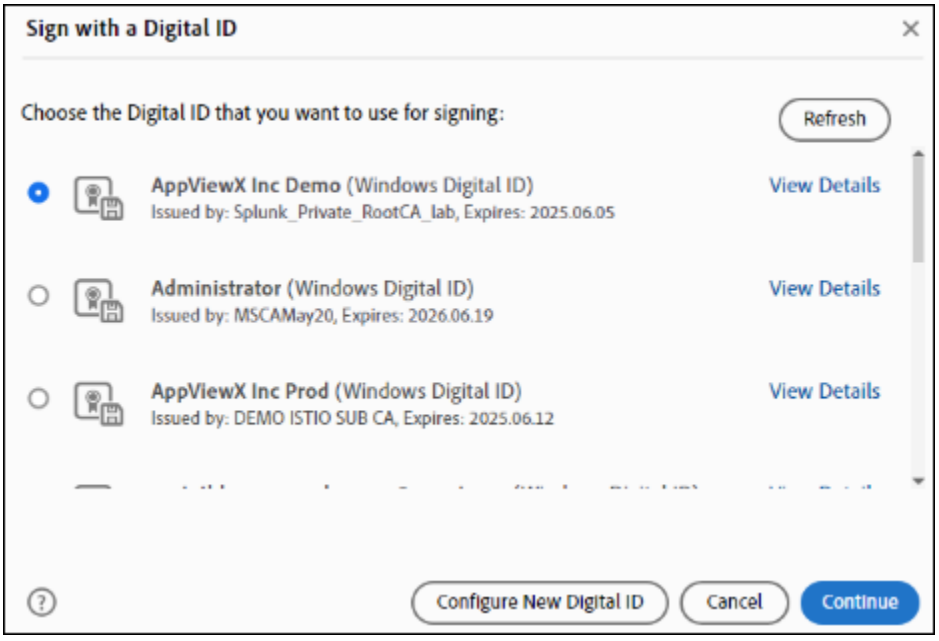
2. Select the **Digitally Sign** option to add a visible digital signature to the document.



3. Use the mouse to draw a designated area for the digital signature anywhere on the document.

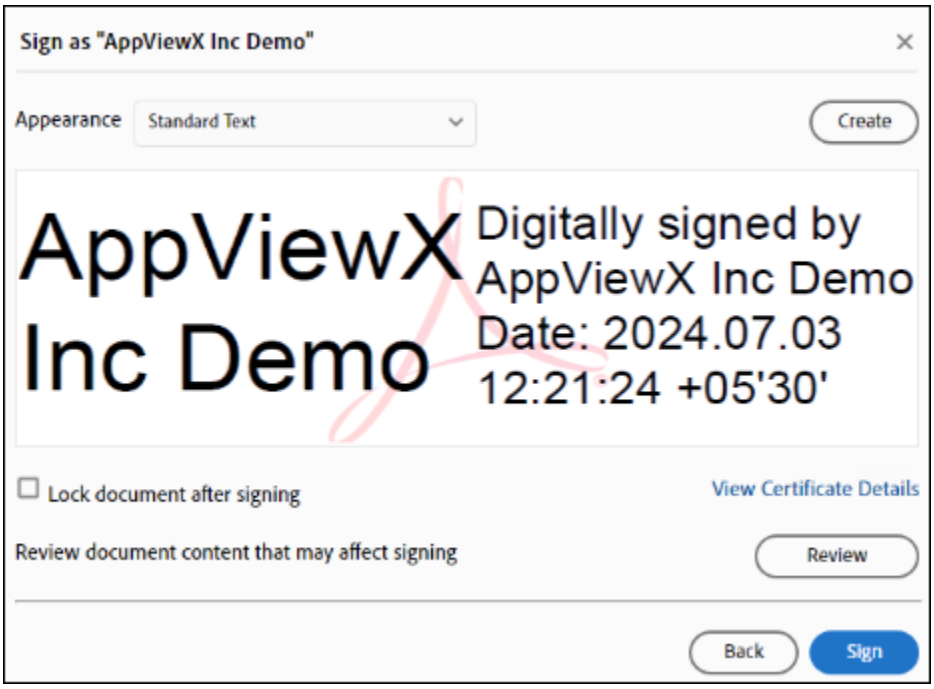


4. Select the appropriate Digital ID (Signing Certificate selected during SIGN+ Package download) from the list of installed code signing certificates in Windows Key Storage. Select the certificate downloaded and installed through SIGN+_Installer and click **Continue**.

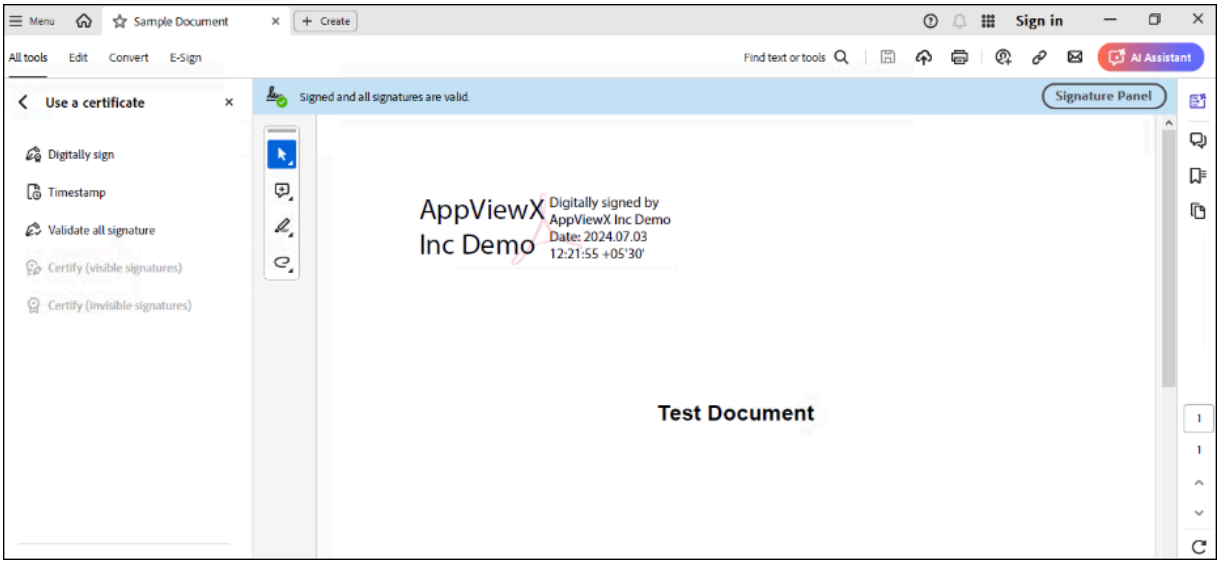


5. Preview or edit the Digital Signature Display information as needed and click “Sign.” To customize the appearance, refer to Adobe’s guide on the Appearance of Digital Signatures.

Note: SHA256 is the default hashing algorithm.

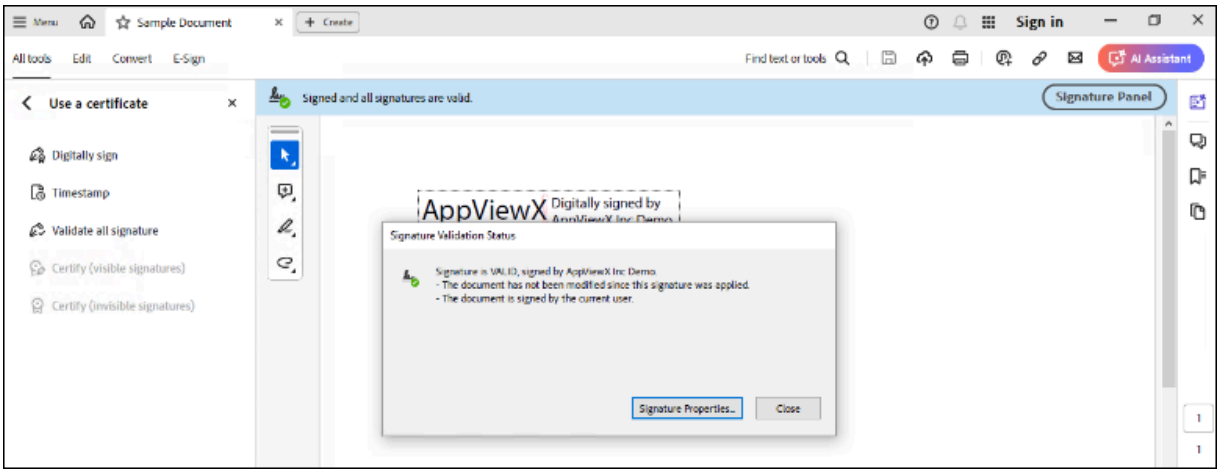


6. Save the signed document to your preferred location.
7. The Digital Signature information will appear in the area specified in **Step 3**.

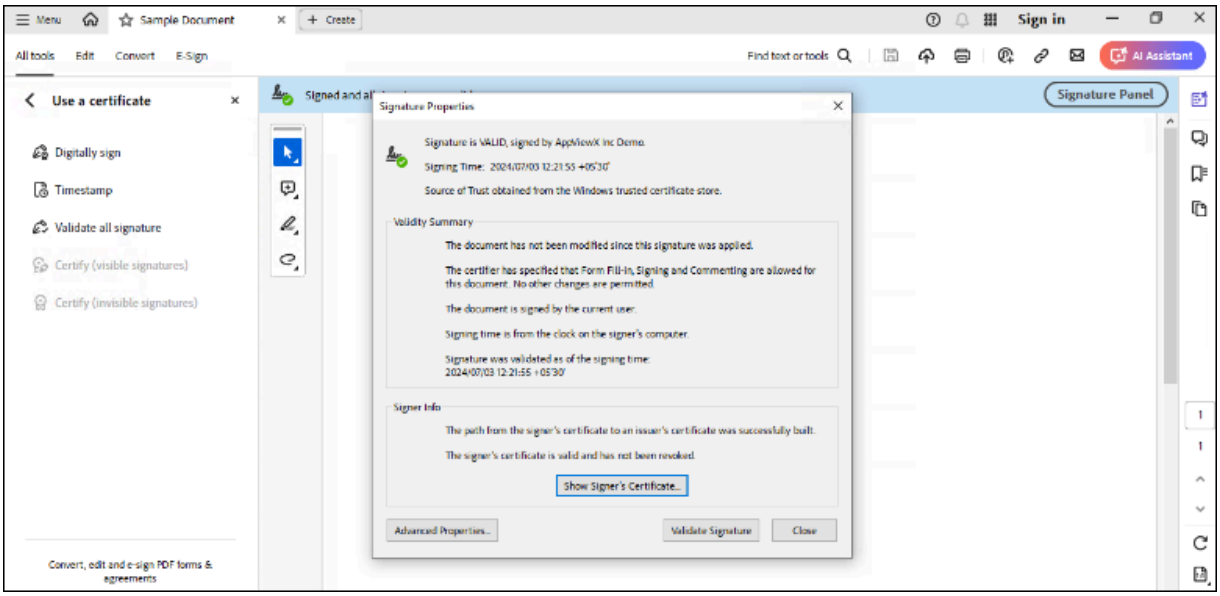


View and Verify Digital Signature

1. Click the digital signature in the document to access the signature details.



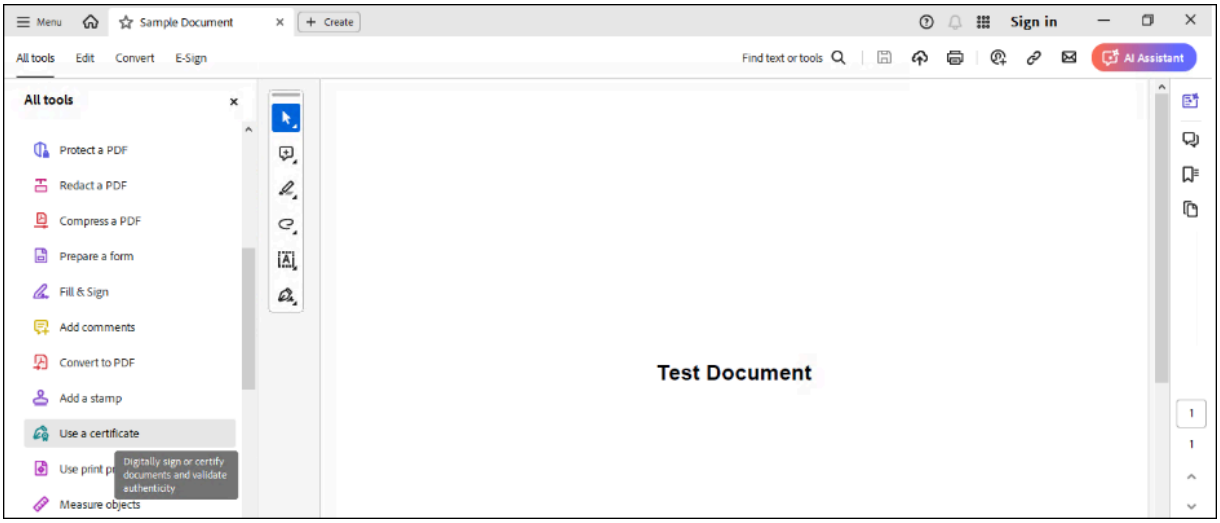
2. View the digital signature properties for detailed information.



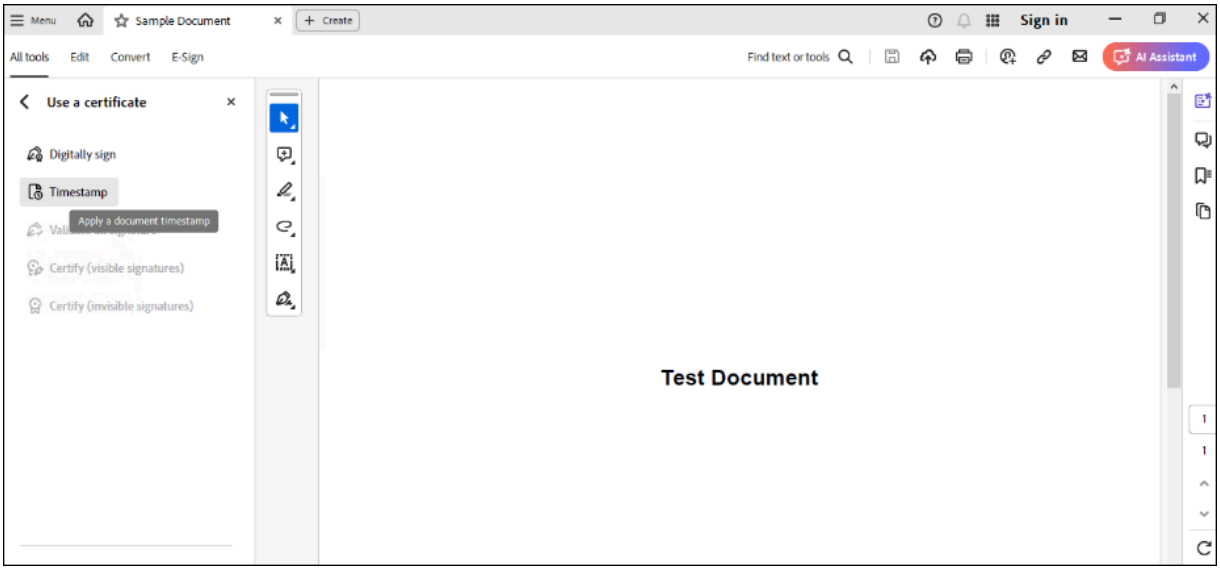
Document Timestamping in Adobe Acrobat

Add Document Timestamping to a PDF

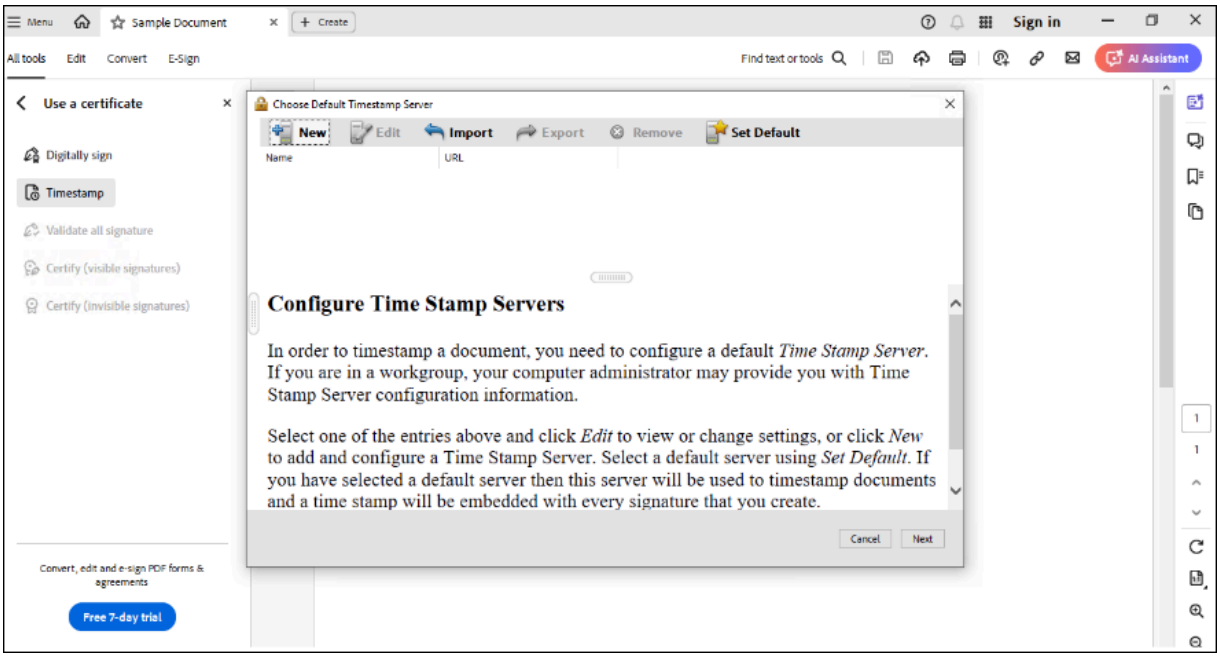
1. Open the PDF to be Timestamped in Adobe Acrobat and go to **All Tools** -> **View More** -> **Use a Certificate** in the Global Bar.



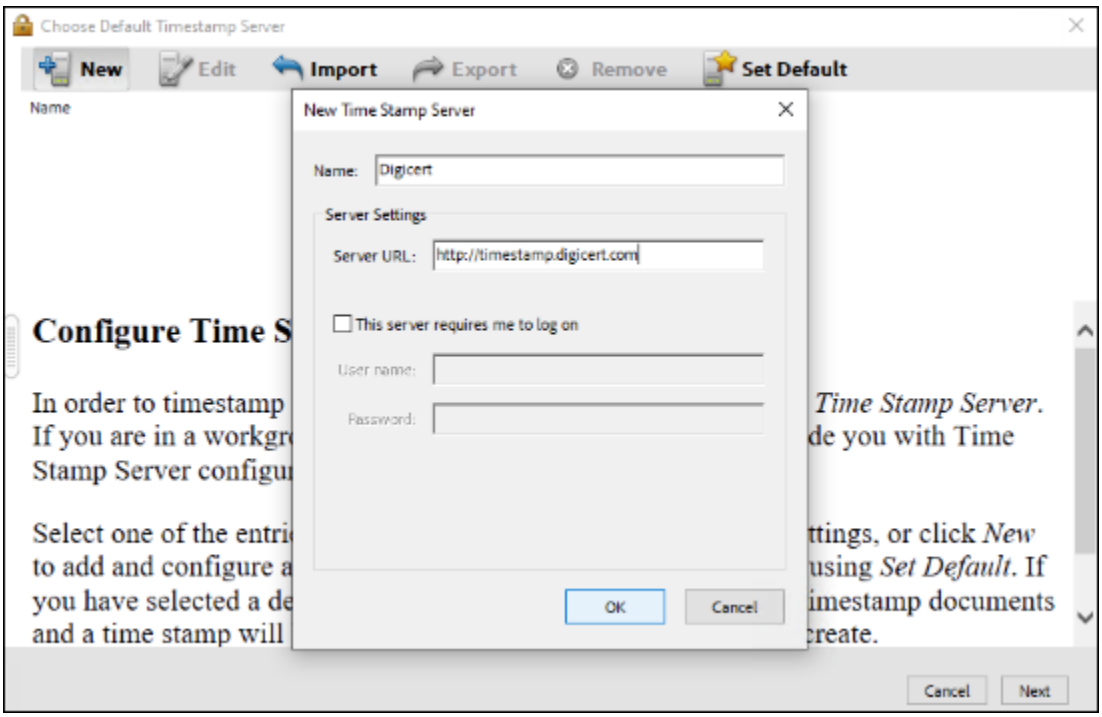
2. Select the **Timestamp** option to apply a document timestamp.



3. Select the **New** option from the prompt to configure the default timestamp server.



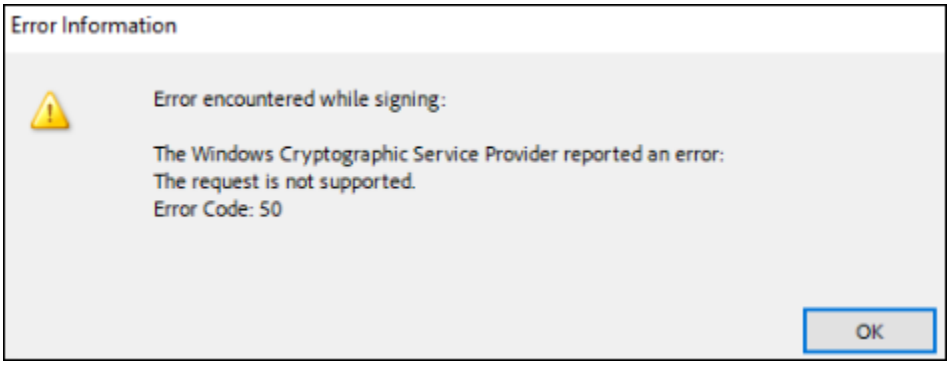
4. Enter the required configurations and click **OK** to set up timestamping.



Troubleshoot Signing Errors

Error Encountered While Signing: The Windows Cryptographic Service Provider Reported an Error.

Error message:



Problem:

This error message can occur due to various reasons, such as connection issues with the server, authentication errors, and other validation messages.

Solution:

For more information on the error message, refer to `AppViewX_CSP_<Day>.log` file in **C:\Users\<username>\AppData\Local\Temp** path.

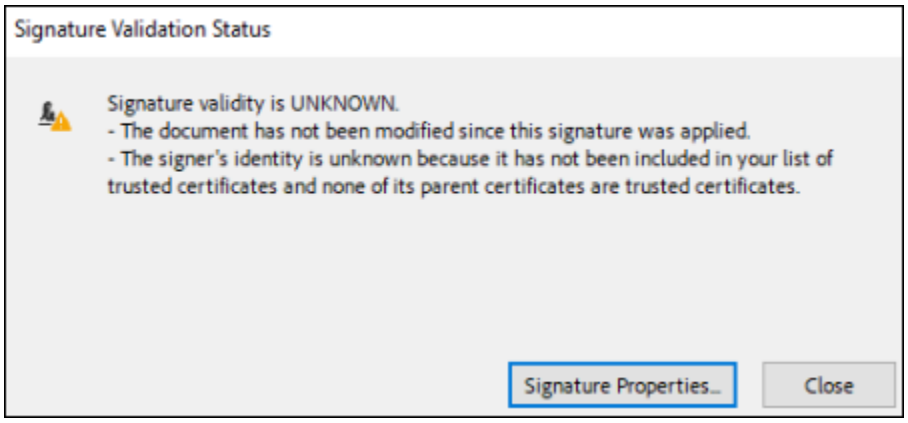
Troubleshoot Signature Verification

At least one signature has problems

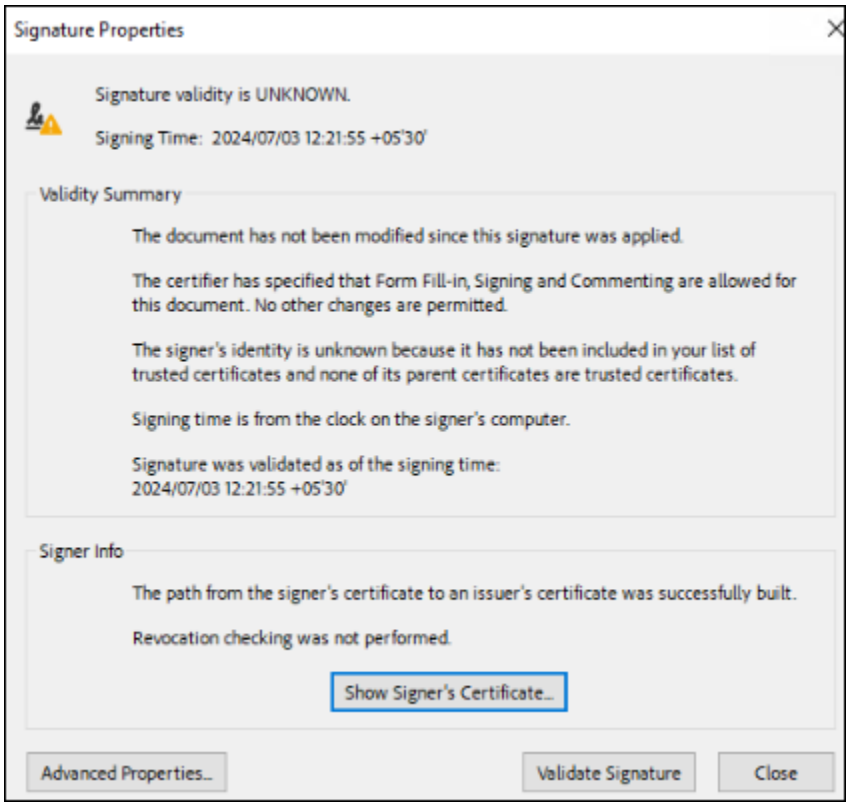
Error message:

- *Signature Banner*

- *Signature Panel*



• *Signature Properties*



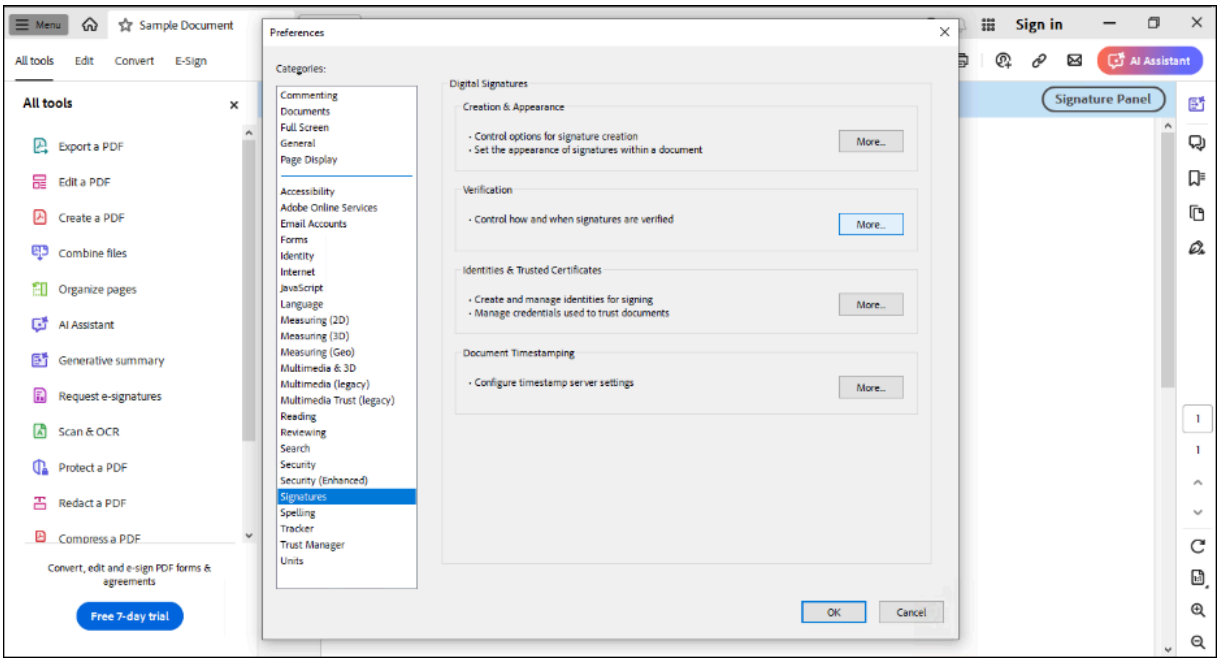
Problem:

This error message occurs when the code signing certificate used for signing is not trusted by Adobe Acrobat.

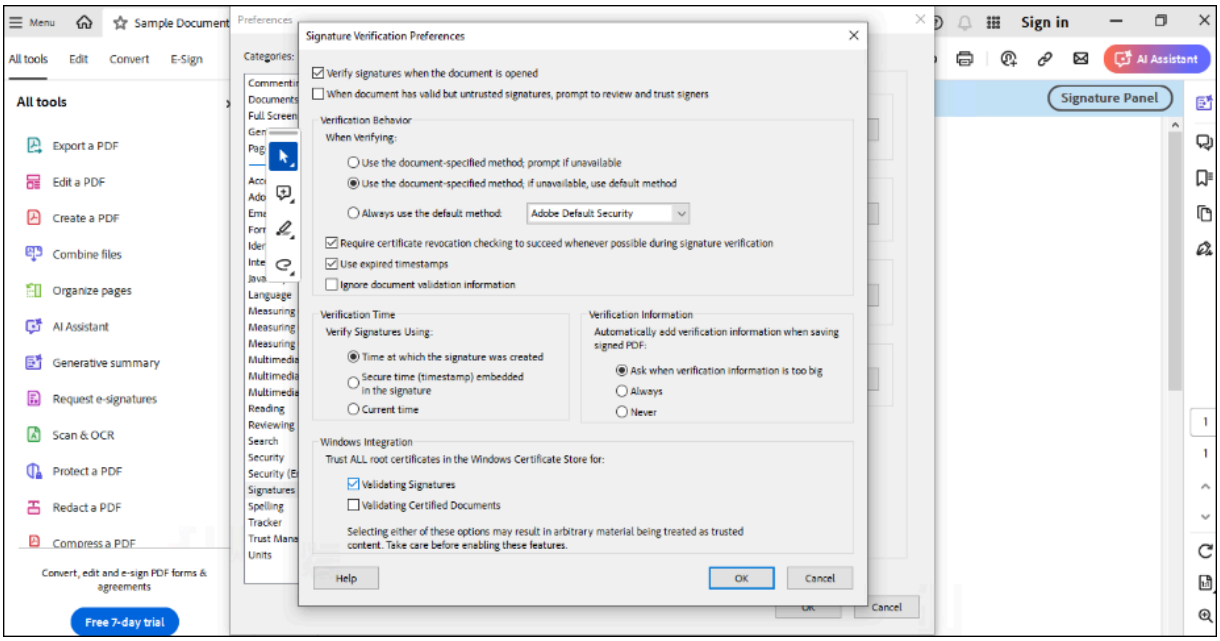
Note: This error message might occur only for Private CA Code Signing Certificates. In case of Public CA Code Signing Certificates, the trust chain would already be present in Adobe.

Solution:

1. Navigate to **Main Menu -> Preferences -> Signatures -> Verifications -> More**.



2. Select the **Validating Signatures** checkbox under **Trust All the root certificates in the Windows Store for**:



Integrating Microsoft Office

Microsoft Office is a suite of software applications developed by Microsoft, designed for creating, editing, and managing documents, presentations, and spreadsheets. It is widely used for personal, educational, and business purposes due to its versatility and user-friendly interface. Microsoft Word, Excel, and PowerPoint provide powerful features for document creation, data analysis, and presentation design, respectively, making them essential tools for enhancing productivity and collaboration across devices and platforms.

Download Microsoft Office

Download and install MS Office for windows from Download Microsoft Office 365 for Windows and Mac.

Sign Office Files with MS Office Using AppViewX CSP

Digital signatures in Microsoft Office documents ensure the authenticity and integrity of files. By applying a digital signature, the document is encrypted with a unique digital certificate that verifies the signer's identity and confirms that the content has not been altered since it was signed. This process enhances security by preventing unauthorized modifications and allows recipients to trust that the document is genuine. Digital signatures in Office applications like Word, Excel, and PowerPoint are essential for maintaining data integrity, particularly in business and legal contexts.

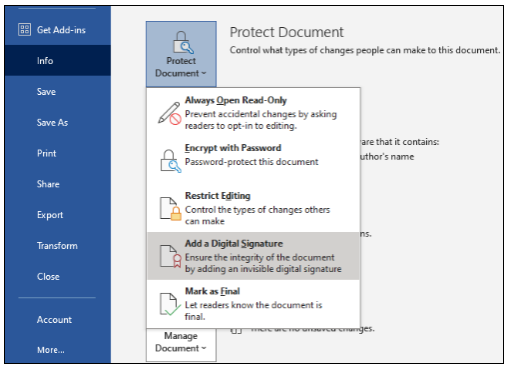
Prerequisites:

1. Execute the AppViewX SIGN+ installer to set up the prerequisites for using AppViewX CSP.
2. Ensure that Microsoft Office Desktop version is installed.

Sign MS Office files

1. Open the document to be signed in Microsoft Word, Excel, or PowerPoint.
2. Click on **File** in the selected application.
3. On the **File** tab, click Info and then
   - **Microsoft Word**

     Click **Protect Document -> Add a Digital Signature**.



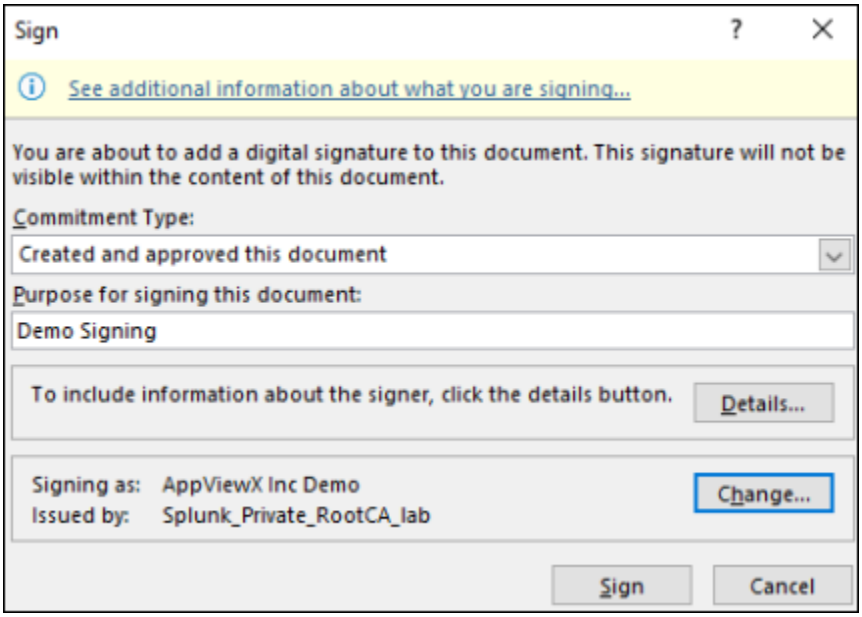
- **Microsoft Excel**

  Click **Protect Workbook -> Add a Digital Signature**.

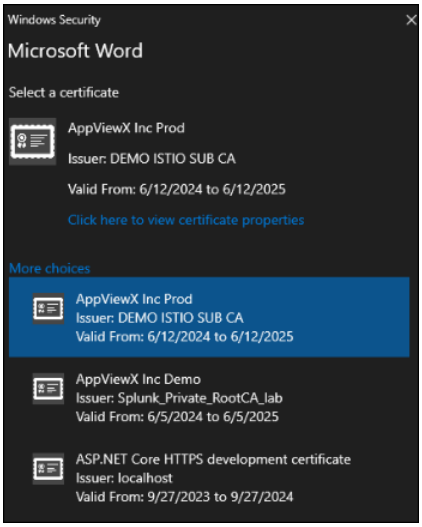
- **Microsoft PowerPoint**

  Click **Protect Presentation -> Add a Digital Signature**.

4. In the **Sign** window, select the commitment type that best represents the signer's role from the **Commitment Type** drop-down list.
   - None
   - Created and approved this document
   - Approved this document
   - Created this document
5. In the **Purpose for signing this document** box, enter the reason for signing.



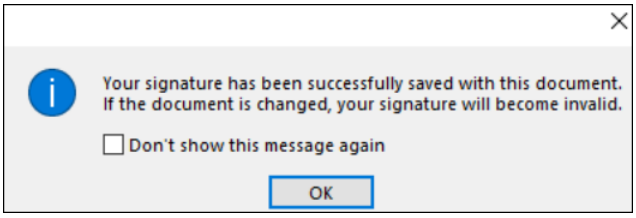
6. To provide signer information, click **Details**. In the Additional Signing Information window, enter the necessary details and click **OK**.
7. Next, click **Change** in the Sign window.
8. In the Windows Security window, select the signing certificate (that was selected while downloading the SIGN+ Package) and click **OK**.



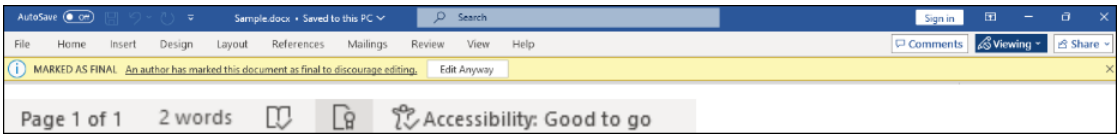
9. In the Sign window, click **Sign**.

> **Note:** SHA256 is the default hashing algorithm.

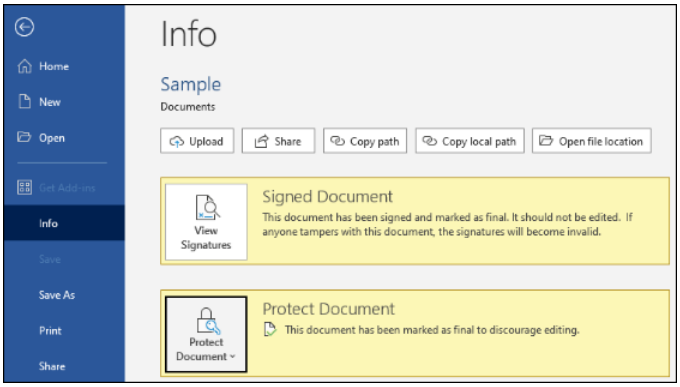
10. In the Signature Confirmation window, read the message and click **OK**.



11. The document is marked as final, and a small ribbon icon appears at the bottom of the document window, indicating it has been signed. If any edits are made, the signature will be removed and must be resigned.



12. To view signer information, click **File -> Info -> View Signatures**.



Troubleshoot Signing Errors

Error Encountered While Signing

Error Message:

MS Office Stopped responding.

Problem:

This error message occurs due to various reasons like error while establishing connection to the server, authentication error and other validation messages.

Solution:

For more information on the error message, refer to `AppViewX_CSP_<Day>.log` file in **C:\Users\<username>\AppData\Local\Temp** path.

Integrating SIGN+ using Native Tools

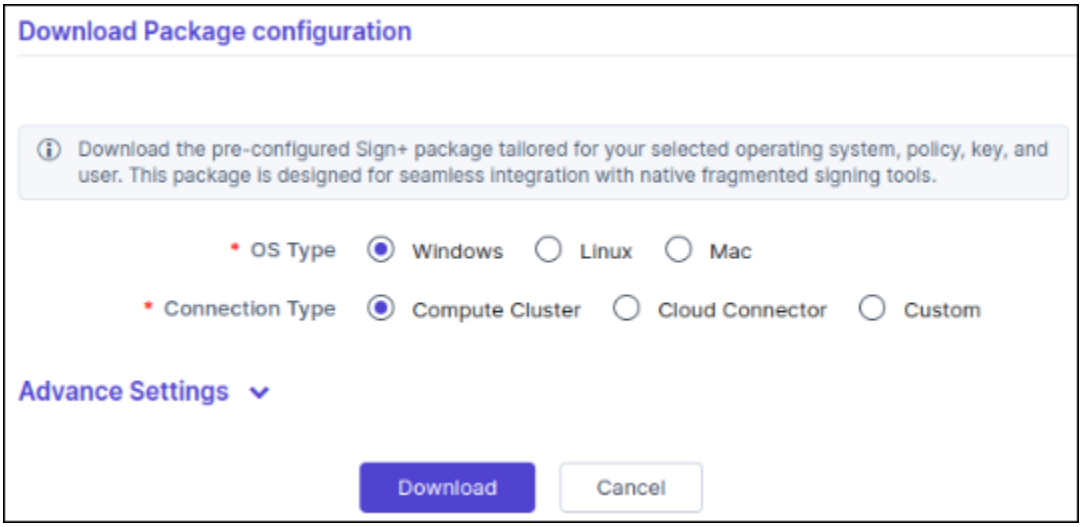
SIGN+ Package

The SIGN+_Package is designed to integrate with the AppViewX SIGN+ Platform for artifact signing. It includes all required libraries and certificates. The package features both the CSP and PKCS#11 library files, facilitating integration with various native signing tools for file signing.

Downloading the SIGN+ Package

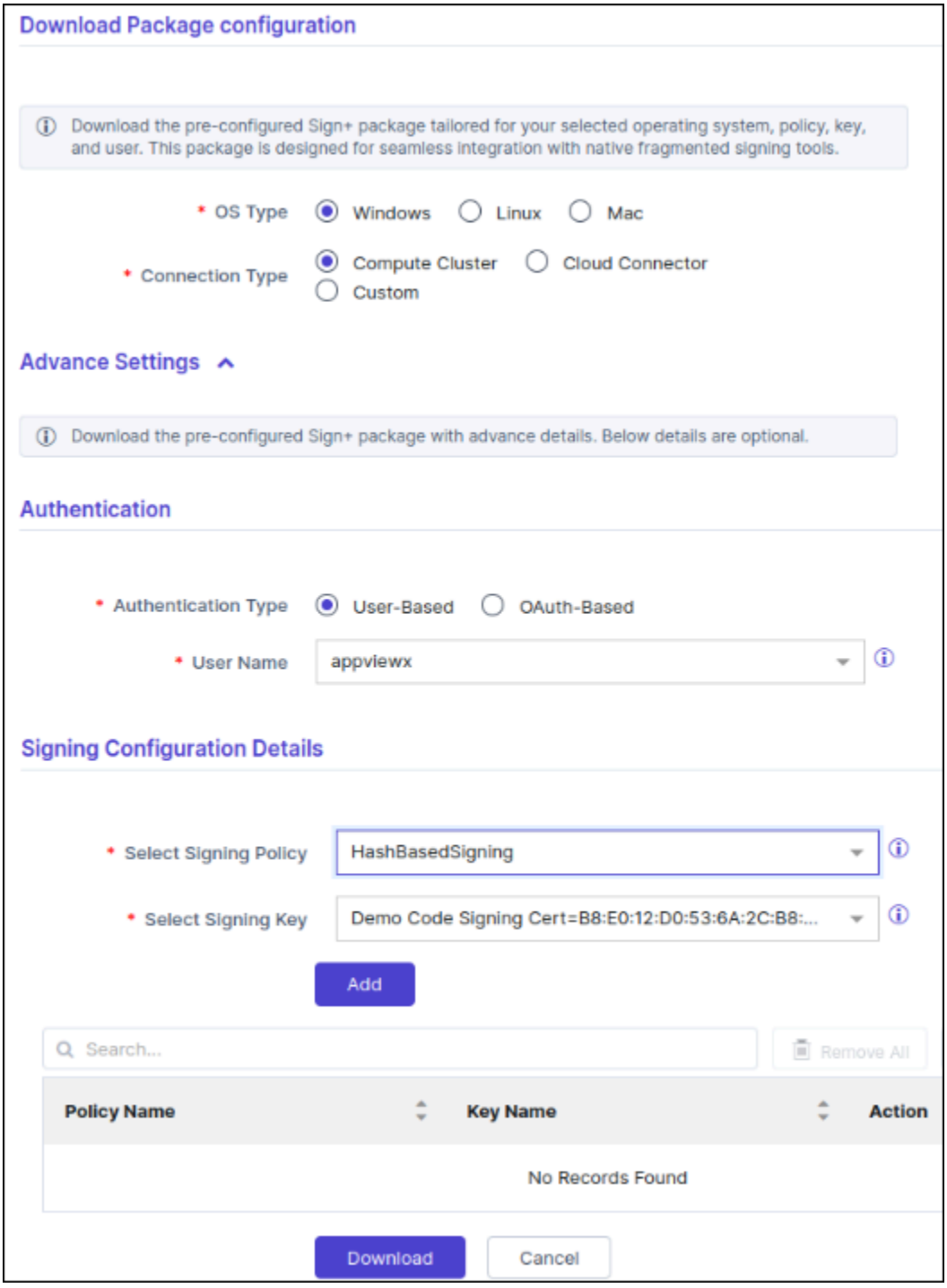
The two available options for downloading the SIGN+_Package are as follows:

- **Basic Package Download**
  - This option downloads only the Base Package, which includes the Installer Executable and SIGN+ API Connector Configurations.
  - Additional configurations, such as Authentication Type, Credentials, and Signing Policy, are fetched at runtime when the SIGN+ Installer is executed.
  - This option is ideal when an Administrator needs to distribute the same SIGN+_Package to a large number of users.



- **Advanced Package Download**

- In this option, all configurations, including Authentication Type, Connector Configurations, Authentication Credentials, and Signing Policy, are selected during the download of the SIGN+ Package.



SIGN+ Installer

AppViewX SIGN+ Installer is a utility executable included in the SIGN+ Package. It handles the installation of prerequisites necessary for using the AppViewX CSP and PKCS#11 Provider with various native signing tools. The installer manages the setup of required configurations and libraries and dynamically generates README files with the necessary commands for use with different tools.

SIGN+ Installer Usage

Note: To ensure seamless signing, it is recommended to avoid adding any other files or folders to the SIGN+ Package.

Windows

The SIGN+ Installer for Windows now provides two installation modes: Admin and Non-Admin.

- **Admin Mode:** Requires administrator privileges and installs dependencies for both AppViewX CSP and the PKCS#11 Provider.
- **Non-Admin Mode:** Does not require administrator privileges and installs only the dependencies needed for the PKCS#11 Provider.

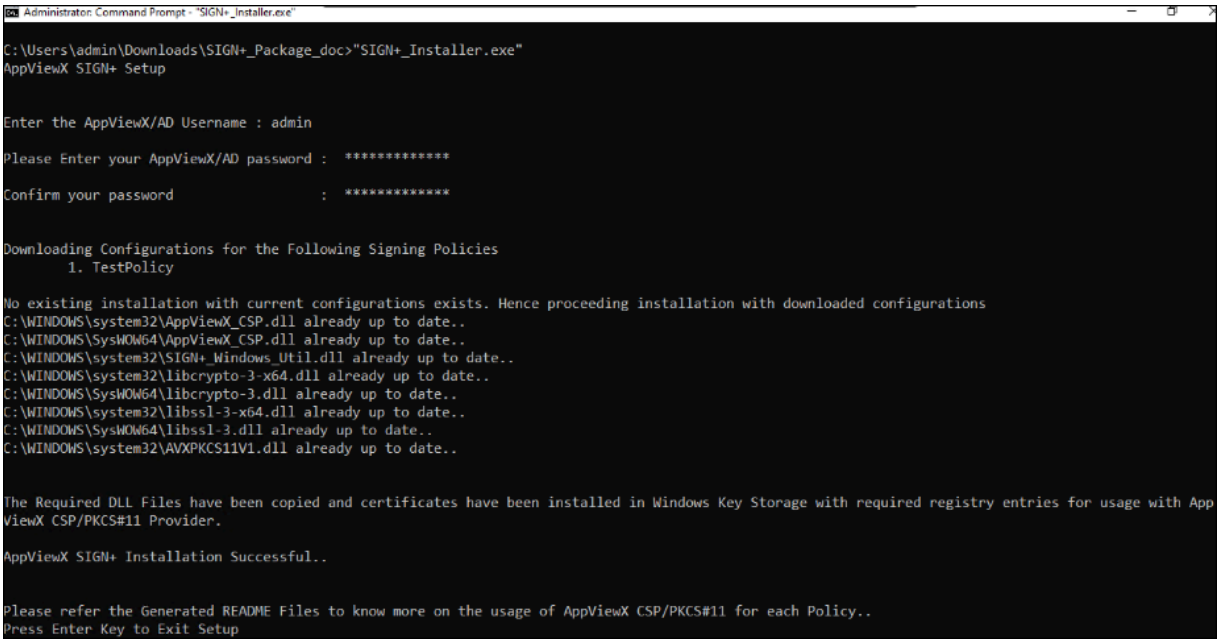
Administrator Mode

Running the SIGN+ Installer Windows executable in Administrator Mode installs the prerequisites and dependencies required for both AppViewX CSP and PKCS#11 Providers. This mode is recommended if signing is performed using Windows Native Tools like signtool, Nuget, Mage, etc.

Note: Administrator privileges are required because setting up the CSP involves copying the DLL file to the System32 directory and making necessary registry entries.

1. Extract the **SIGN+_Package.zip** and open the extracted folder.
2. Run **SIGN+_Installer.exe** as an Administrator.

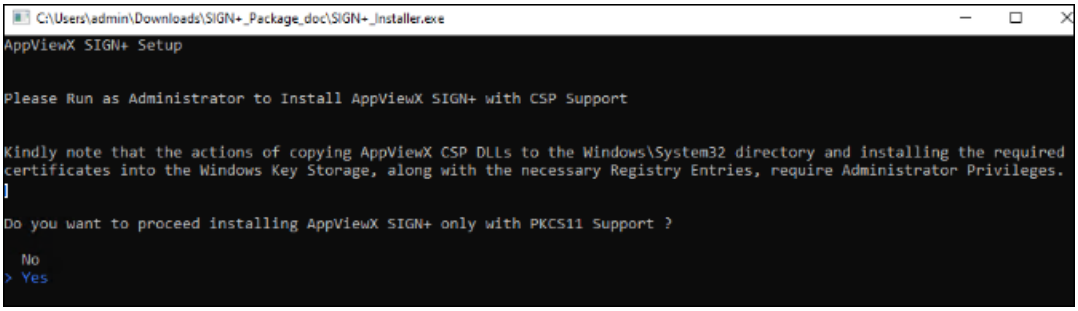
3. Enter the requested details to install the prerequisites for using the AppViewX CSP/PKCS#11 Provider with native signing tools.



Non Administrator Mode

Running the SIGN+ Installer Windows executable in Non-Administrator Mode installs the prerequisites and dependencies required only for the AppViewX PKCS#11 Provider. This mode is recommended if there are restrictions on running in Administrator mode on the signing machine, and signing is performed using Java-based tools like Jsign, Jarsigner, Apksigner, etc.

1. Extract the **SIGN+_Package.zip** and open the extracted folder.
2. Run **SIGN+_Installer.exe**.
3. Choose **Yes** when prompted for **“Do you want to proceed installing AppViewX SIGN+ only with PKCS11 Support?”**



4. Enter the requested details to install the prerequisites for using the AppViewX PKCS#11 Provider with native signing tools.

Linux

1. Extract the **SIGN+_Package.zip** and open the extracted folder.
2. Use the following command to provide execution permissions to the SIGN+_Installer executable file.

```
chmod +x <path_to_SIGN+_Installer>
```

3. Execute the **SIGN+_Installer** executable using the following command.

```
./<path_to_SIGN+_Installer>
```

4. Enter the requested details to install the prerequisites for using the PKCS#11 Provider with native signing tools.

MacOS

1. Extract the **SIGN+_Package.zip** and open the extracted folder.
2. Use the following command to provide execution permissions to the SIGN+_Installer executable file.

```
chmod +x <path_to_SIGN+_Installer>
```

3. Execute the **SIGN+_Installer** executable using the following command.

```
./<path_to_SIGN+_Installer>
```

> **Note:** If the executable is blocked from running on macOS, go to **System Settings > Privacy & Security > Security**, locate **SIGN+_Installer**, and click "**Allow Anyway**" to proceed with execution.

4. Enter the requested details to install the prerequisites for using the PKCS#11 Provider with native signing tools.

> **Note:** The PKCS#11 library for macOS is compiled for **x64 architecture** to maintain backward compatibility. Therefore, when using PKCS#11 with other tools on ARM-based Macs (M-series), make sure to download and use the **x64 versions** of the respective tools (e.g., Jarsigner, Java) for signing.

SIGN Installer Functionalities

The SIGN+_Installer includes functionality that allows you to upgrade your existing installation with newly configured Signing Policies and Libraries. This eliminates the need to download a new SIGN+_Package. The available functionalities are listed below:

SIGN+_Installer Help

The SIGN+_Installer help command lists down all the supported functionalities of the SIGN+_Installer. For more information on a specific command use ***SIGN+_Installer help <command>***

Sample Output:

```
"SIGN+_Installer.exe" help

Available commands:

    - install: Usage: SIGN+_Installer Install --authtype <basic|oauth> --username <username> --password <password> --overwriteInstallation

    - updatecredentials: Usage: SIGN+_Installer UpdateCredentials

    - updateconnectorurl: Usage: SIGN+_Installer UpdateConnectorURL

    - upgrade: Usage: SIGN+_Installer Upgrade --binary --certs --both --overwriteInstallation

    - print: Usage: SIGN+_Installer Print

    - uninstall: Usage: SIGN+_Installer Uninstall

    - help: Usage: SIGN+_Installer Help <command>
```

```
"SIGN+_Installer.exe" help install

Usage: SIGN+_Installer Install --authtype <basic|oauth> --username <username> --password <password> --overwriteInstallation

Install AppViewX SIGN+ with the specified options.


Options:

  --authtype: Specify Basic Authentication or oAuth Based Authentication

  --username: The username or clientId used for authentication.

  --password: The password or clientSecret used for authentication.

  --overwriteInstallation: Force or overwrite the existing installation.
```

SIGN+_Installer Install - Usage

The SIGN+_Installer install command is used to install the SIGN+_Package without user interaction. This is helpful for non-interactive shell environments such as CICD Pipeline Environments.

Sample Usage:

```
"SIGN+_Installer.exe" install --authtype basic --username “user” --password "password" --overwriteInstallation
```

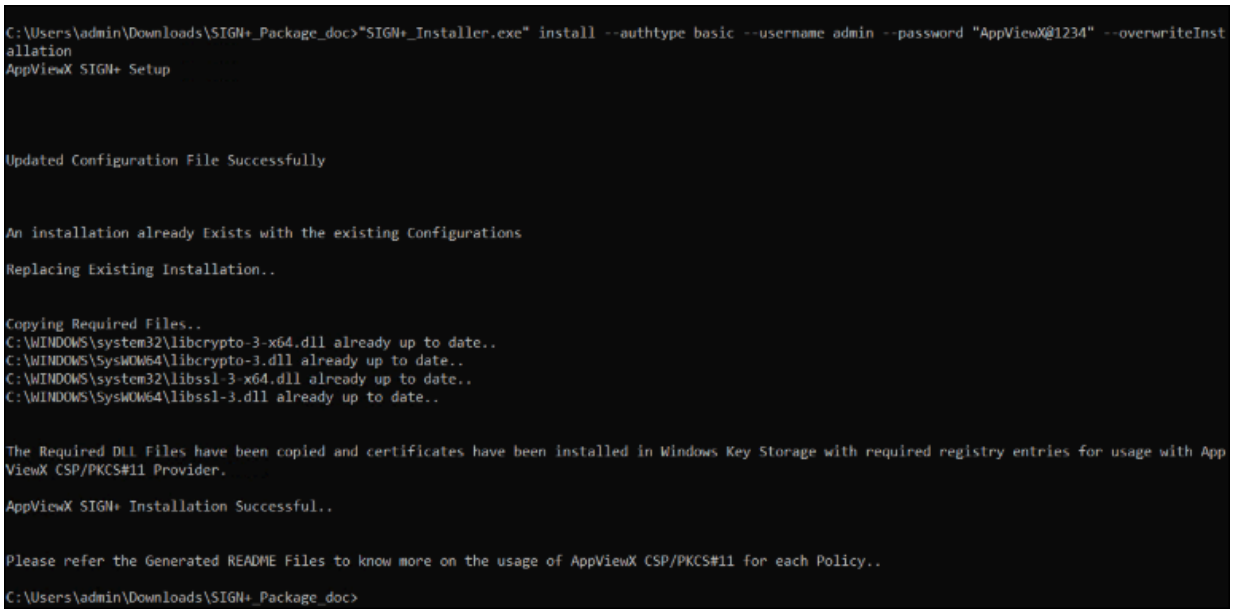
Parameters explanation:

- ***- - authtype -*** Specified the type of authentication to be used. Accepts the values “basic” for Username/ Password based authentication and “oauth” for Service Account Based Authentication.
- ***- - username -*** Accepts the Username or clientId used for authentication.
- ***- - password -*** Accepts the Password or clientSecret used for authentication.

- ***- - overwriteInstallation -*** Specify this parameter to overwrite the existing installation with new configurations from the SIGN+ Server.

> **Note:** In Linux or MacOS, it is recommended to provide username or password arguments in single quotes if it contains any special characters. This will prevent the misinterpretation of special characters (e.g., the dollar sign '$') as argument parameters.

Sample Usage:

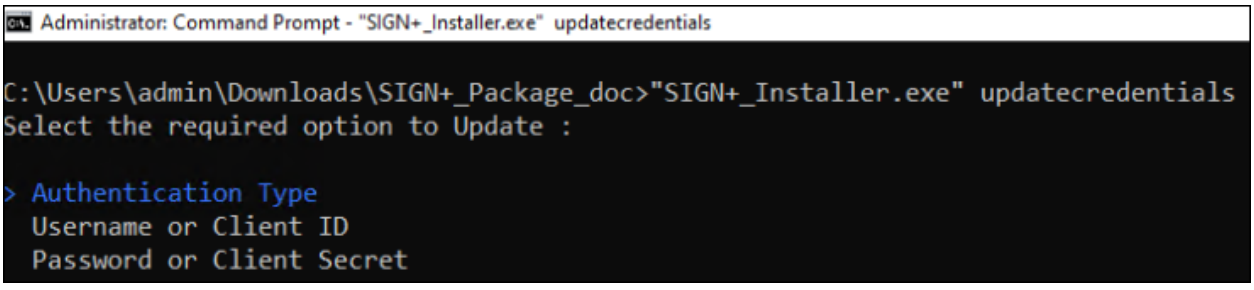


SIGN+_Installer UpdateCredentials

This option allows the user to update the authentication type and credentials. This can be helpful in cases where the password or client secret is expired or has to be changed post installation.

The above option allows the user to update 3 options as follows:

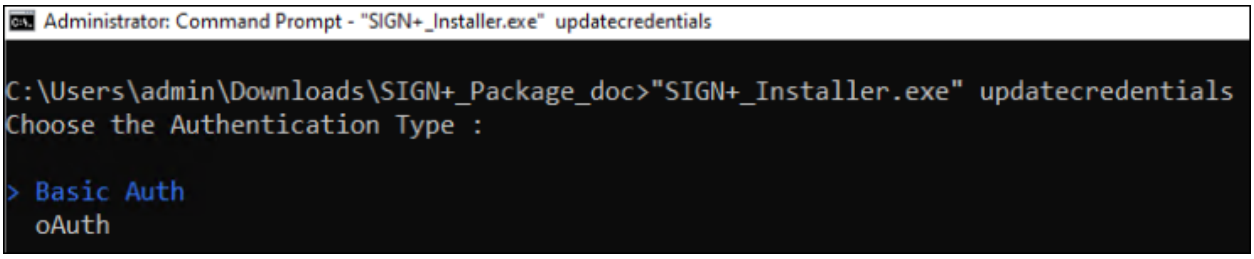
```
"SIGN+_Installer.exe" updatecredentials
```

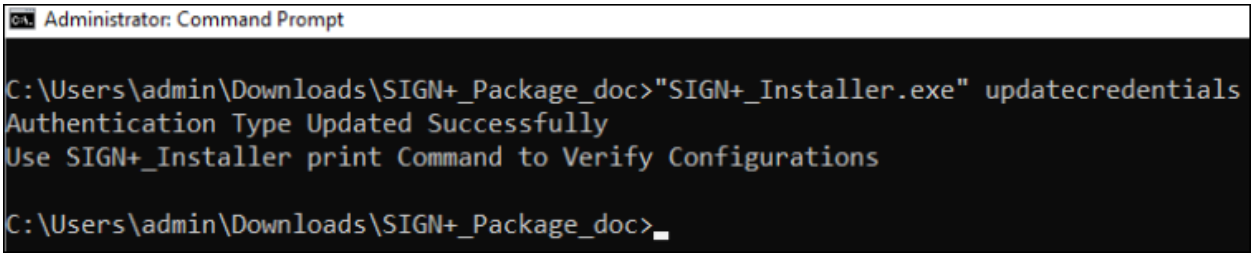


Updating the Authentication type

Select the required Authentication Type to be used by selecting the required option using the Arrow Keys.

Sample Usage:

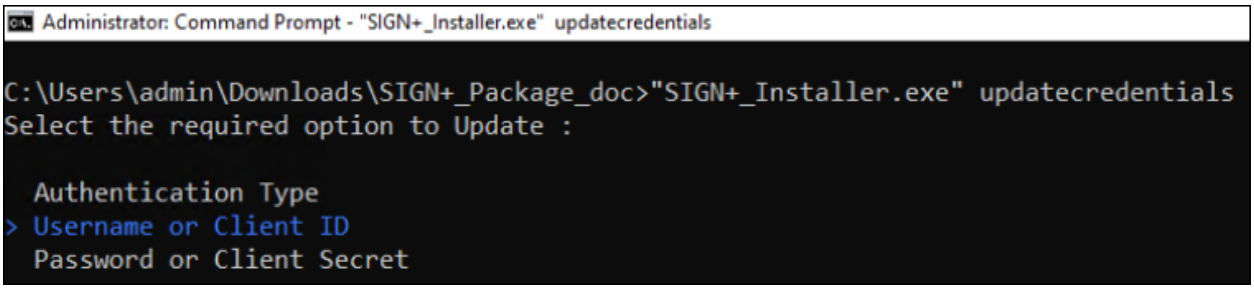


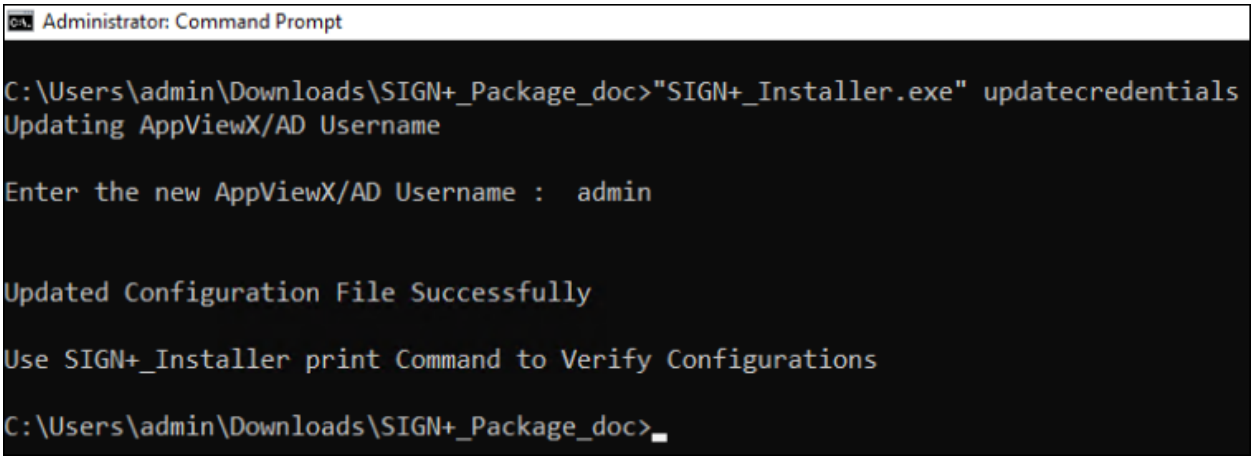


Updating Username or Client ID

This option can be selected to update the Username or Client ID used for Authentication

Sample Usage:

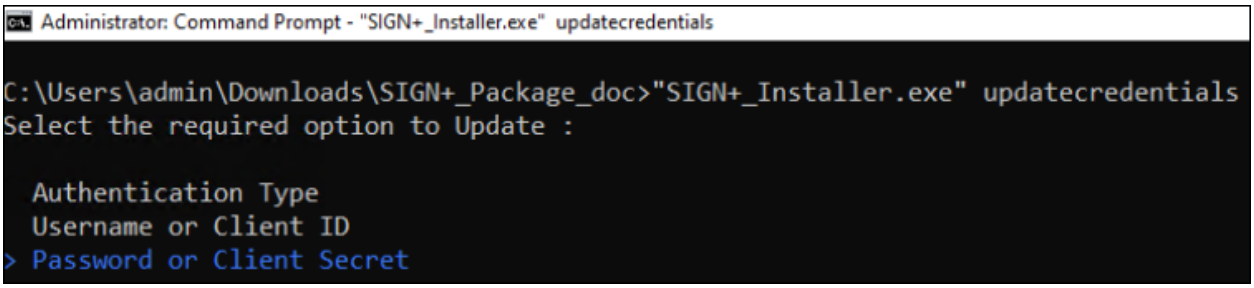


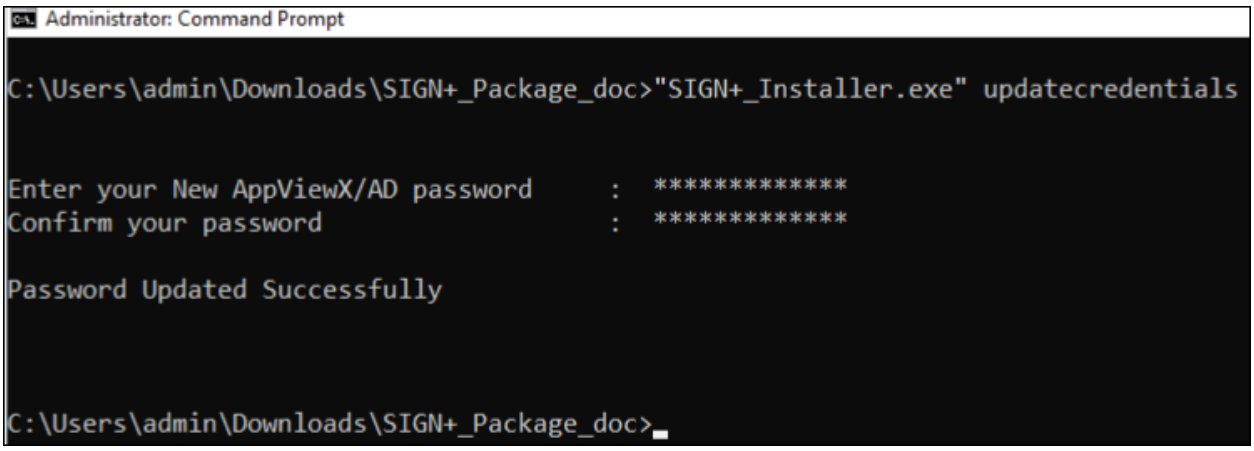


Updating Password or Client Secret

This option can be selected to update the Password or Client Secret used for Authentication.

Sample Usage:



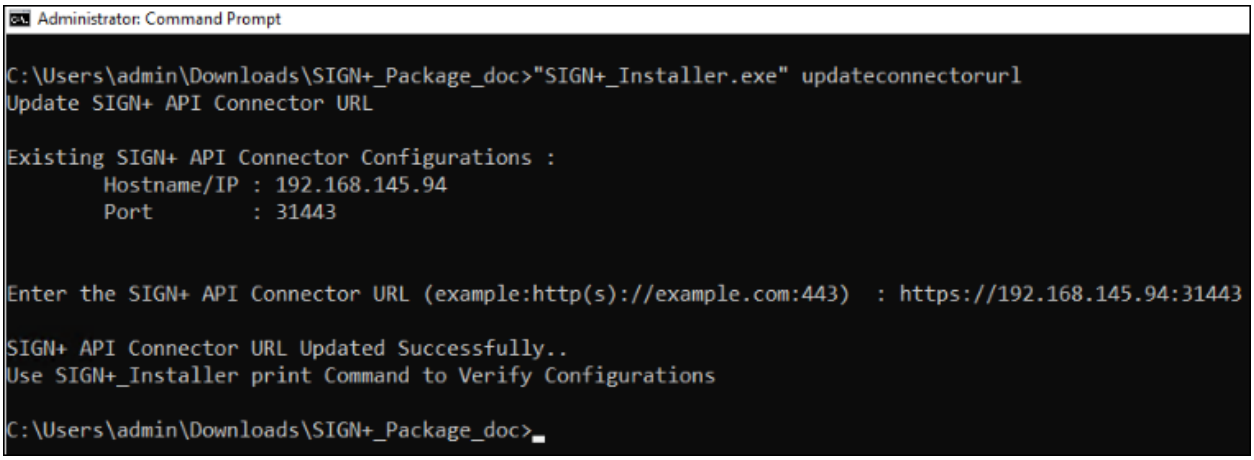


SIGN+_Installer UpdateConnectorUrl

This option can be used in cases where the SIGN+ API Connector URL has to be changed post installation. This is helpful in cases if the package was downloaded by selecting the cloud connector option but it has to be updated to a different URL like the Load Balancer URL

Sample Usage:

```
"SIGN+_Installer.exe" updateconnectorurl
```



SIGN+_Installer Upgrade

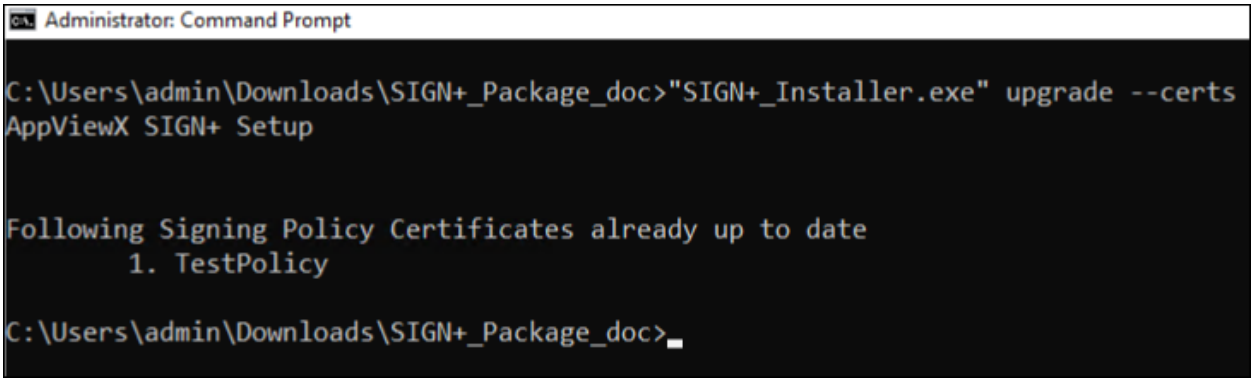
This option can be used in cases where the existing installation has to be upgraded with new SIGN+ Policy Configurations or libraries or both without the need to download and install a new SIGN+ Package.

Upgrade Certificates

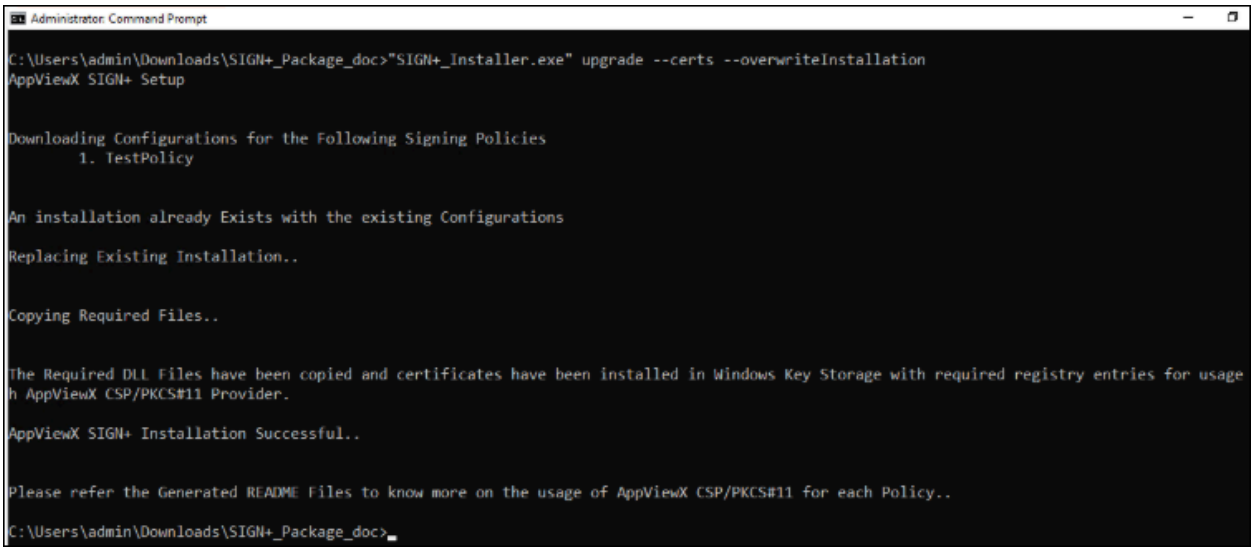
This option can be used to upgrade the existing installation with the newly configured policies configured in the SIGN+ Server. This will automatically download and install the certificates and policy configurations of any modified or newly added policy in the server and install in the local machine. The corresponding README Files will also be generated.

Sample Usage:

```
"SIGN+_Installer.exe" upgrade --certs
```




```
"SIGN+_Installer.exe" upgrade --certs --overwriteInstallation
```



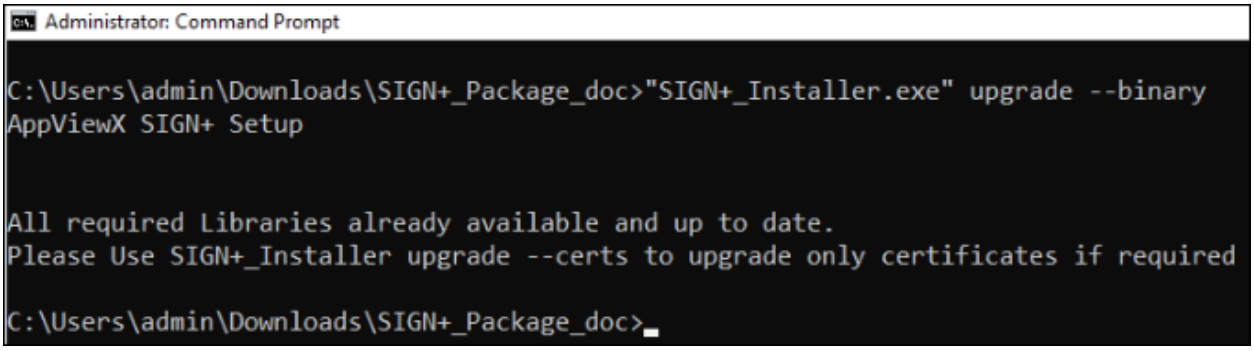
Note: It is recommended to use the overwriteInstallation flag to download all the Signing Policy configurations like timestamping url and hashing algorithm and install in local. If the flag is not given, only the certificates are upgraded.

Upgrade Binary

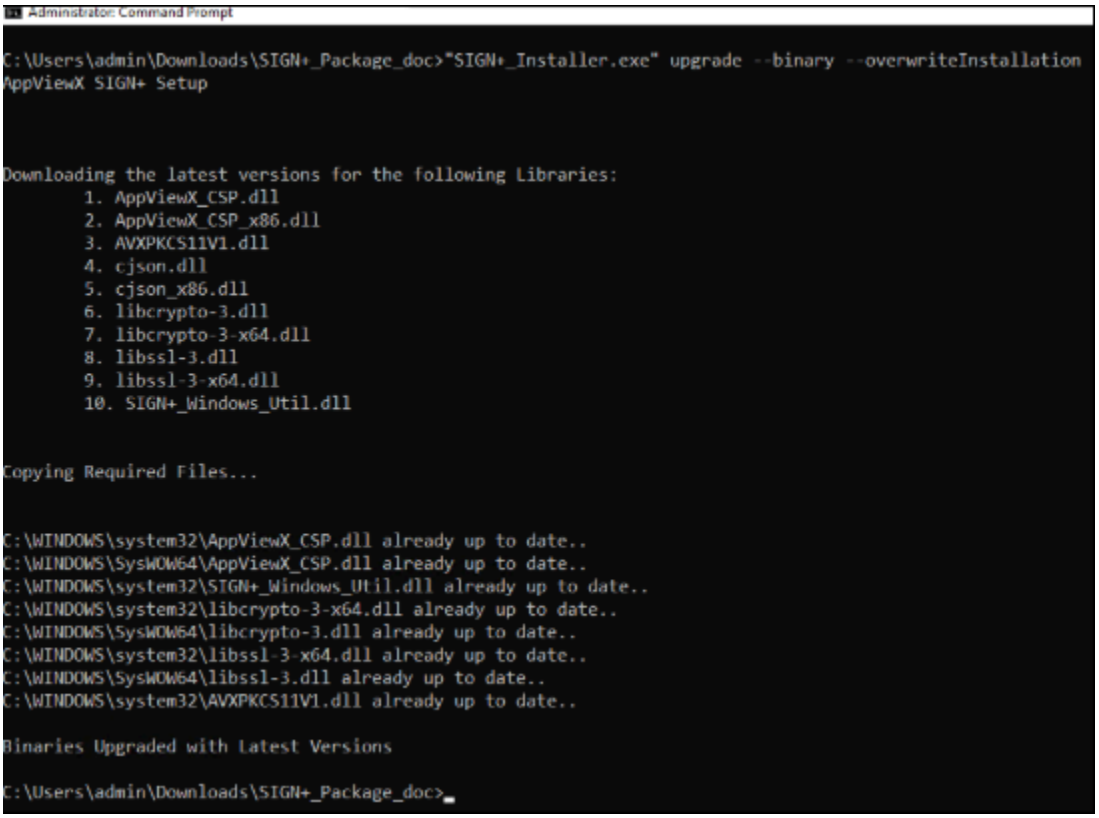
This option can be used to upgrade the existing installation with the latest version of the AppViewX CSP and PKCS#11 Library Files.

Sample Usage:

```
"SIGN+_Installer.exe" upgrade --binary
```



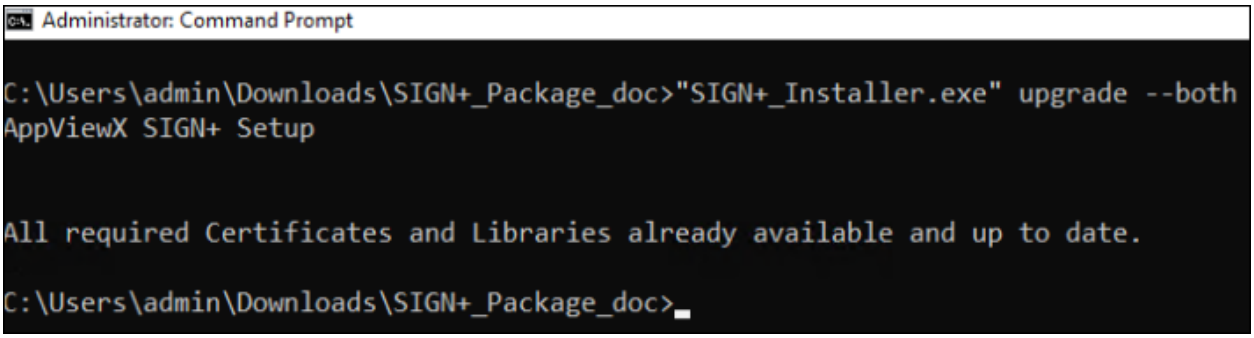

```
"SIGN+_Installer.exe" upgrade --binary --overwriteInstallation
```



Upgrade Both

Use this option to upgrade both the certificates and library files with the latest version.

```
"SIGN+_Installer.exe" upgrade --both
```




```
"SIGN+_Installer.exe" upgrade --both --overwriteInstallation
```

```
C:\Users\admin\Downloads\SIGN+_Package_doc>"SIGN+_Installer.exe" upgrade --both --overwriteInstallation
AppViewX SIGN+ Setup


Downloading Configurations for the Following Signing Policies
        1. TestPolicy

Downloading the latest versions for the following Libraries:
        1. AppViewX_CSP.dll
        2. AppViewX_CSP_x86.dll
        3. AVXPKCS11V1.dll
        4. cjson.dll
        5. cjson_x86.dll
        6. libcrypto-3.dll
        7. libcrypto-3-x64.dll
        8. libssl-3.dll
        9. libssl-3-x64.dll
        10. SIGN+_Windows_Util.dll



An installation already Exists with the existing Configurations

Replacing Existing Installation..


Copying Required Files..
C:\WINDOWS\system32\AppViewX_CSP.dll already up to date..
C:\WINDOWS\SysWOW64\AppViewX_CSP.dll already up to date..
C:\WINDOWS\system32\SIGN+_Windows_Util.dll already up to date..
C:\WINDOWS\system32\libcrypto-3-x64.dll already up to date..
C:\WINDOWS\SysWOW64\libcrypto-3.dll already up to date..
C:\WINDOWS\system32\libssl-3-x64.dll already up to date..
```

SIGN+_Installer Print

Use this option to print the configurations of the current SIGN+ installation.

```
"SIGN+_Installer.exe" print
```

```
C:\Users\admin\Downloads\SIGN+_Package_doc>"SIGN+_Installer.exe" print


API Connector Configurations
        Authentication Type       :  basicAuth
        Username/Client ID        :  admin
        Connector Hostname/IP     :  192.168.145.94
        Connector Port            :  31443


Signing Policy Configurations
        Policy Name       : TestPolicy
        Signing Key       : Demo Code Signing Cert=0A:D1:6D:1D:C3:61:2E:D0:FB:AE:E8:6A:9E:B8:F1
        Hashing Algorithm : SHA-384
        Timestamping URL  : http://timestamp.digicert.com
```

SIGN+_Installer UpdateRetryTimeout

Use this option to update the retry timeout configuration value (in seconds) which will be used in AppViewX CSP/PKCS#11 Libraries in case of any timeout errors from the server side.

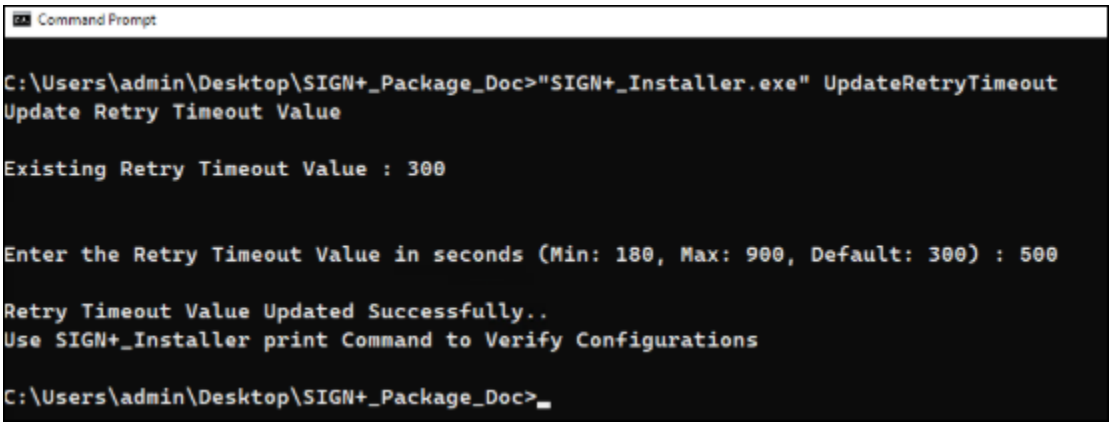
```
"SIGN+_Installer.exe" UpdateRetryTimeout
```

An additional parameter “retryTimeout” is added in the avx_sign_config.json file. This additional configurable timeout value is consumed by the AppViewX CSP and PKCS#11 Libraries with a maximum of 3 retries

The minimum supported value is 180

The maximum supported value is 900

The default value is 300.



SIGN+_Installer Uninstall

The Uninstall option in the SIGN+_Installer executable can be used to clean up the SIGN+ installation by removing the library files, configuration files, and log files.

Sample Command and Output

```
"SIGN+_Installer.exe" Uninstall

AppViewX SIGN+ Uninstaller




Deleting Configuration Files..

Deleting Library Files..

Deleting Temporary Log Files..


Uninstallation Successful


Press Enter key to exit...
```

- Signtool
- JARSigner
- APKSigner
- JSign
- NuGet
- Esptool
- XMLSecTool
- Mage
- Set-AuthenticodeSignature
- Cosign
- OpenSSL
- Troubleshooting Guide for SIGN+ Native Tools Integration

Signtool

SignTool is a command-line tool included with Microsoft Visual Studio and the Windows Software Development Kit (SDK). It is primarily used for code-signing Windows executables and files to indicate their authenticity and origin. SignTool allows developers and software publishers to sign their software with digital signatures, which can be used to verify the integrity and source of the files.

- Common Use Cases for SignTool
- File types that can be signed using SignTool
- Download SignTool
- Setting the PATH Environment Variable
- Sign Authenticode Files with Signtool using AppViewX CSP
- Sign Excel Macros with Signtool using AppViewX CSP

Common Use Cases for SignTool

- **Code Signing:** SignTool is often used to digitally sign executable files (such as .exe and .dll) and script files (like .msi, .cab, and .ps1) on the Windows platform.
- **Driver Signing:** Device drivers for Windows must be signed with a digital signature to ensure compatibility and security.

- **Timestamping:** SignTool can apply timestamp signatures to files, ensuring the signature remains valid even after the certificate has expired.
- **Verification:** SignTool can verify the digital signatures of files to confirm their authenticity.

File types that can be signed using SignTool

- **SignTool (64-bit):**

  .appx, .appxbundle, .arx, .cab, .cat, .cbx, .cpl, .crx, .dbx, .deploy, .dll, .drx, .efi, .exe, .js, .msi, .msix, .msixbundle, .msm, .msp, .ocx, .psi, .psm1, .stl, .sys, .vbs, .vsix, .vxd, .wsf, .xap, .xsn

- **SignTool (32-bit):**

  .doc, .docm, .dot, .dotm, .mpp, .mpt, .pot, .potm, .ppa, .ppam, .pps, .ppsm, .ppsm, .ppt, .pptm, .pub, .vdw*, .vdx*, .vsd*, .vsdm, .vss*, .vssm, .vst*, .vstm, .vsx*, .vtx*, .wiz*, .xla, .xlam, .xls, .xlsb, .xlsm, .xlt, .xltm

Download SignTool

SignTool is included in the Windows Software Development Kit (SDK). To install it:

- Download the Windows SDK.
- Run the winsdksetup.exe file.
- Follow the wizard's instructions to complete the installation.
- SignTool (64-bit) is located in:

```
C:\Program Files (x86)\Windows Kits\10\bin\<version>\x64
```

- SignTool (32-bit) is located in:

```
C:\Program Files (x86)\Windows Kits\10\bin\<version>\x86
```

Setting the PATH Environment Variable

Operating systems use the environment variable called PATH to determine where executable files are stored on your system. You can use the PATH environment variable to store the file path to your signing tools to ensure that the command-line interface (CLI) can reference these signing tools.

To set the path to your signing tools via the command line:

```
set PATH=%path%;<path_to_signing_tool_folder>
```

Command Sample:

```
set PATH=%path%;C:\Program Files (x86)\Windows Kits\10\bin\<version>\x64\
```

To set the path to your signing tools for your system or account:

1. Search for "environment variables" in the Windows start menu.
2. Select "Edit environment variables for your account" or "Edit system environment variables."
3. In the "Environment Variables" window, locate the "Path" variable under "System Variables" or "User Variables."
4. Double-click on the "Path" variable.
5. Click **New**.
6. Select **Browse**.
7. Navigate to the path where the signing tool is located. For example: C:\Program Files (x86)\Windows Kits\10\bin\<version>\x64\
8. Click **OK** to save the path.
9. Click **OK** again to close the "Environment Variables" dialog.

By following these steps, you'll set the PATH environment variable to include the folder containing signtool.exe, ensuring that the command line can access the signing tools.

Sign Authenticode Files with Signtool using AppViewX CSP

Prerequisites

- Run the AppViewX SIGN+ Installer to install the prerequisites required to use the AppViewX CSP with Signtool.

Signing with SignTool

Command:

```
signtool.exe sign /f <path to certificate>  /fd <digest algorithm>  /csp <csp_name> /k <key_alias_name> /tr <timestamp_url> /td <timestamp digest algorithm>

<input_file_path>
```

- **/f <path to certificate>:** Path to your code-signing certificate.
- **/fd <digest algorithm>:** Specifies the hashing algorithm.
- **/csp <csp_name>:** Name of Cryptographic Service Provider (CSP).
- **/k <key_alias_name>:** Key Container Name.
- **/tr <timestamp_url>:** Provides a timestamp from a trusted timestamping authority.

- **/tr <timestamp_digest>:** Specifies the timestamping Digest algorithm.
- **<input_file_path>:** Path to the file to be signed.

The **<path to certificate>, <digest algorithm>, <csp_name>, <key_alias_name>, <timestamp_url>, <timestamp_digest>** parameters are auto generated based on the signing policy configurations in the README after running the SIGN+ Installer.

Verification with SignTool

Command:

```
signtool.exe verify /v /pa <input_file_path>
```

Sign Excel Macros with Signtool using AppViewX CSP

Prerequisites

- Download and install Microsoft Office Subject Interface Packages for Digitally Signing VBA Projects
- Download and install Visual C++ 2010
- Run the AppViewX SIGN+ Installer to install the prerequisites required to use the AppViewX CSP with Signtool.

Set up macro signing

Once you install all required tools, open a command prompt in Administrator mode. Next, run the commands:

```
regsvr32.exe <complete path to msosip.dll>
```

```
regsvr32.exe <complete path to msosipx.dll>
```

If successful, you will see a message: **"DllRegister Server in <complete file path> succeeded."**

Signing with SignTool

Use the SignTool present in the path C:\Program Files (x86)\Windows Kits\10\bin\<version>\x86 to sign Excel macros. To sign, use the command:

```
<path_to_32bit_signtool.exe> sign /f <path to certificate>  /fd <digest algorithm>  /csp <csp_name> /k <key_alias_name> /tr <timestamp_url> /td <timestamp digest algorithm>  <input_file_path>
```

- **/f <path to certificate>:** Path to your code-signing certificate.
- **/fd <digest algorithm>:** Specifies the hashing algorithm.

- **/csp <csp_name>:** Name of Cryptographic Service Provider (CSP).
- **/k <key_alias_name>:** Key Container Name.
- **/tr <timestamp_url>:** Provides a timestamp from a trusted timestamping authority.
- **/tr <timestamp_digest>:** Specifies the timestamping Digest algorithm.
- **<input_file_path>:** Path to the file to be signed.

The **<path to certificate>, <digest algorithm>, <csp_name>, <key_alias_name>, <timestamp_url>, <timestamp_digest>** parameters are auto generated based on the signing policy configurations in the README after running the SIGN+ Installer.

Verification with SignTool

Command:

```
<path_to_32bit_signtool.exe> verify /v /pa <input_file_path>
```

JARSigner

The Java Development Kit (JDK) provides the JarSigner tool, which developers use to sign Java Archive (JAR) files and other Java-related files, including Java Web Start applications and Java applets. It is primarily used to verify the authenticity and integrity of Java applications and libraries. JarSigner adds digital signatures to JAR files, which allows users and systems to confirm that the files have not been tampered with since they were signed and that they come from a trusted source.

Here are some key features and use cases of JarSigner:

- **Code Integrity:** JarSigner is used to sign Java applications and libraries to ensure their code integrity. When users or systems run a signed JAR file, the Java Runtime Environment (JRE) can verify the digital signature to ensure that the code has not been altered since it was signed.
- **Authentication:** JarSigner helps establish the authenticity of the software publisher. By signing JAR files with a digital certificate issued by a trusted certificate authority (CA), software publishers can prove their identity to users and systems.
- **Java Web Start:** JarSigner is commonly used with Java Web Start applications. When users launch a Java Web Start application, the JRE checks the digital signature of the JAR files it downloads to ensure their validity and security.

- **Java Applets:** JarSigner can also be used to sign Java applets, which are small Java applications that run within web browsers. This allows web browsers to verify the applet's source and integrity.
- **Timestamping:** JarSigner can add timestamp information to the digital signature. Timestamping ensures that the signature remains valid even after the certificate used for signing has expired. This is particularly important for long-lived applications.

To use JarSigner, you typically need a code-signing certificate issued by a CA. You then use JarSigner to apply the digital signature to the JAR files. When distributing Java applications, especially those that users will download from the internet, signing with JarSigner is an important security practice to establish trust.

JarSigner is a command-line tool, and its usage involves specifying the JAR file to be signed, the digital certificate to use, and other optional parameters such as timestamping. Once the JAR file is signed, it can be distributed to users with confidence in its authenticity and integrity.

Sign with Jarsigner

Use Jarsigner to sign, timestamp, and verify the following file types:

- **.ear**
- **.jar**
- **.sar**
- **.war**
- **.zip**

Install Jarsigner

Windows

To download the JDK from Oracle:

1. Navigate to Oracle > JDK 11 > Windows.
2. Download the **x64 MSI** installer.

3. Run the jdk-11_windows_x64_bin.msi file that was downloaded.
4. Follow the instructions in the wizard to complete the installation.

   Jarsigner.exe should be located in the file path: C:\Program Files\Java\jdk-11\bin.

   Alternatively, you can download and install the JDK from OpenJDK.

Linux

On Debian and Ubuntu Linux distributions:

1. Open the Terminal application.
2. To install Java, run:

```
sudo apt install -y default-jdk
```

3. To verify the Java version, run:

```
java --version
```

On RHEL, CentOS, and Fedora Linux distributions:

1. Open the Terminal application.
2. To install Java, run:

```
yum install java-1.8.0-openjdk.x86_64
```

3. To verify the Java version, run:

```
java -version
```

Set the PATH environment variable

Operating systems use the environment variable called PATH to determine where executable files are stored on your system. Use the PATH environment variable to store the file path to your signing tools to ensure that the CLI can reference these signing tools.

Windows

You can configure the signing tools using the command line or environment variables.

To set the path to your signing tools via the command line:

```
set PATH=%path%;<path to signing tool folder>
```

To set the path to your signing tools for your system or account:

1. Search for environment variables in the Windows start menu.
2. Select Edit environment variables for your account or Edit system environment variables.
3. Double-click on the Path variable.
4. Click **New**.
5. Select Browse.
6. Select the path to the signing tool.
7. Click **OK** to save the path.
8. Click **OK** to close the dialog box.

Linux

To set the path to your signing tools via the command line:

1. Launch the Terminal application.
2. Open the file in an editor:

```
nano ~/.profile
```

3. Add any export definitions you need:

```
export PATH=<Path to Jarsigner>
```

4. Click **CTRL+X** to exit.
5. Click **Y** to save.
6. Click Enter to keep the same file name.
7. Execute the new .profile by restarting Terminal or using:

```
source ~/.profile
```

Sign JAR Files with JARSigner using AppViewX CSP/PKCS#11 Provider

Prerequisites

- Run the AppViewX SIGN+ Installer to install the prerequisites required to use the AppViewX CSP/PKCS#11 Provider with Jarsigner.

AppViewX CSP with JarSigner:

```
jarsigner.exe -verbose -storetype "Windows-My" -keyStore NONE -tsa <time_stamp_url> <input_file_path> -signedjar <output_file_path> -sigalg <signature algorithm> <keypair alias>
```

The **<time_stamp_url>**, **<signature algorithm>** and **<keypair alias>** parameters are auto generated in the README after running the SIGN+ Installer.

AppViewX PKCS#11 Provider with JarSigner:

```
jarsigner.exe -verbose -keystore NONE -storetype PKCS11 -certs -providerclass sun.security.pkcs11.SunPKCS11 -providerArg <path to AVXPKCS11V1.cfg>
 <input_file_path> -signedjar <output_file_path> -tsa <time_stamp_url> -sigalg <signature algorithm> <keypairalias>
```

The **<path to AVXPKCS11V1.cfg>**, **<time_stamp_url>**, **<signature algorithm>** and **<keypair alias>** parameters are auto generated in the README after running the SIGN+ Installer.

Verify Signed Artifact with JarSigner

The following command can be used to verify signed artifact with JarSigner:

```
jarsigner.exe -verify -verbose <input_file_path>
```

APKSigner

APKSigner is a tool commonly used in Android app development to sign Android application packages (APK files). Signing APK files is a critical step in the Android app development process, as it ensures the authenticity and integrity of the app. Here's what you need to know about APKSigner:

- Purpose of APK Signing
- Key Points about APKSigner
- Files that can be signed with Apksigner and PKCS11
- Installation of Apksigner
- Sign APK Files with APKSigner using AppViewX PKCS#11 Provider

Purpose of APK Signing

Authentication: When you sign an APK file, you attach a digital signature to it using a cryptographic key. This signature serves as proof that the app has not been tampered with and that it comes from a trusted source. Users and Android devices use this signature to verify the app's legitimacy.

Integrity: The signature also ensures the integrity of the APK. If any changes are made to the APK after it is signed, the signature becomes invalid. This prevents malicious parties from modifying the app's code or resources.

Updates: When you release updates to your app, you must sign them with the same key as the original APK. This allows users to update the app without losing their data or settings.

Key Points about APKSigner

Command-Line Tool: APKSigner is typically used as a command-line tool or JAR file, and it's available for Windows, macOS, and Linux.

Signing Key: To sign APK files, you need a signing key, which includes a private key for signing and a corresponding public key for verifying the signature. It is crucial to protect your signing key because it's the foundation of your app's trustworthiness.

Google Play Store: If you intend to publish your app on the Google Play Store, you'll need to sign it with a key. Google Play uses the app's signature to verify updates and maintain a consistent identity for the app.

Signing Process: The process of signing an APK involves running the APKSigner tool with the appropriate command-line arguments, specifying the input APK file, the signing key, and the output file. APKSigner handles the signing and generates a signed APK.

Files that can be signed with Apksigner and PKCS11

- .aab (Android App Bundle)
- .apk (Android Application Package)

Installation of Apksigner

Download **apksigner.jar** from GitHub or install using Android Studio using the following steps:

To download the Android SDK and install Apksigner, follow these steps:

- Download Android Studio from the official website.
- Run the android-studio file that was downloaded.
- Follow the steps in the Android Studio Setup wizard to complete the installation.
- Launch Android Studio and complete the setup wizard.

Apksigner JAR should now be available in the file path:

For Windows: C:\Users\<username>\AppData\Local\Android\Sdk\build-tools\33.0.2\lib

Sign APK Files with APKSigner using AppViewX PKCS#11 Provider

Prerequisites

- Run the AppViewX SIGN+ Installer to install the prerequisites required to use the AppViewX PKCS#11 Provider with APKSigner.

Signing with Apksigner

Command:

```
java -jar <path_to_apk_signer_jar> sign --provider-class sun.security.pkcs11.SunPKCS11 --provider-arg <path to AVXPKCS11V1.cfg> --ks NONE
 --ks-type PKCS11 --ks-pass pass:12345678 --ks-key-alias <keypair alias> --in "<input_file_path>" --out "<output_file_path>" --v1-signing-enabled false
 --v2-signing-enabled false --v3-signing-enabled true --v4-signing-enabled false
```

The **<path to AVXPKCS11V1.cfg>**, **<keypair alias>** parameters are auto generated based on the signing policy configurations in the README after running the SIGN+ Installer.

Verifying the Signature with Apksigner

Command:

```
java -jar <path_to_apk_signer_jar> verify -verbose --print-certs <input_file_path>
```

JSign

Jsign is a Java implementation of Microsoft Authenticode that offers platform-independent signing of executable files for Windows, including various file types such as .appx, .exe, .msi, and more. It serves as an alternative to native signing tools like SignTool on Windows or Mono development tools on Unix systems.

- Sign with Jsign
- Installation of Jsign
- Sign Authenticode Files with JSign using AppViewX PKCS#11 Provider

Sign with Jsign

Jsign can be used to sign the following file types:

.appx, .appxbundle, .arx, .cab, .cat, .cbx, .cpl, .crx, .dbx, .deploy, .dll, .drx, .efi, .exe, .js, .msi, .msix, .msixbundle,

.msm, .msp, .ocx, .ps1, .psm1, .stl, .sys, .vbs, .vxd, .wsf, .xap, .xlsm, .xsn

Installation of Jsign

1. Download **jsign-5.0.jar** from GitHub.
2. Move the folder to the location of your choice.

Sign Authenticode Files with JSign using AppViewX PKCS#11 Provider

Prerequisites

- Run the AppViewX SIGN+ Installer to install the prerequisites required to use the AppViewX PKCS#11 Provider with JSign.

Signing with Jsign

Command:

```
java -jar <path_to_jsign_jar> --keystore <path to AVXPKCS11V1.cfg> --storetype PKCS11 --storepass 12345678 --alias <keypair alias>  --alg <digest algorithm>
 --tsaurl <timestamp url> <input_file_path>
```

The **<path to AVXPKCS11V1.cfg>**, **<keypair alias>**, **<digest algorithm>**, **<timestamp url>** parameters are auto generated based on the signing policy configurations in the README after running the SIGN+ Installer.

NuGet

NuGet is a Command Line Interface (CLI) that provides functionality to install, create, publish, and manage packages without making any changes to project files.

Sign with NuGet

Use NuGet to sign .nupkg files.

Download NuGet

1. Download nuget.exe from NuGet Gallery | Downloads.
2. Move nuget.exe to your preferred file path.

Set PATH environment variable (Optional)

Operating systems use the environment variable PATH to determine where executable files are stored on your system. Use the PATH environment variable to store the file path to your signing tools to ensure that the CLI can reference these signing tools.

You can set the PATH environment variable to the folder that contains nuget.exe using the command line or environment variables.

To set the path to your signing tools via command line:

```
set PATH=%path%;<path to signing tool folder>
```

Command sample:

```
set PATH=%path%,C:\Program Files (x86)\
```

To set the path to your signing tools for your system or account:

1. Search for environment variables in the Windows start menu.
2. Select Edit environment variables for your account or system environment variables.
3. Double-click on the Path variable.
4. Click New
5. Select Browse.
6. Select the path to the signing tool. **Example:** C:\Program Files (x86)\Nuget
7. To save the path, click OK.
8. To close the dialog box, click OK.

Sign Windows packages with NuGet using AppViewX CSP

NuGet is a package manager for .NET development that allows you to publish, share, and consume reusable code packages. NuGet is used to sign packages to provide an additional layer of trust and security when distributing software libraries and components. Most importantly, NuGet maintains a reference list of packages used in a project and the ability to restore and update those packages from that list.

Prerequisites:

1. Run the AppViewX SIGN+ Installer to install the prerequisites to use the AppViewX CSP.
2. Installed nuget.exe

Install sample NuGet package

This creates a directory with the name HelloWorld.

```
nuget install HelloWorld
```

By default, all packages installed from the NuGet package manager are signed by the repository. You can verify the package.

Verify a Nuget Package

```
nuget verify -All HelloWorld.1.3.0.17\*
```

Sign a Nuget Package

To sign using a certificate fingerprint:

```
nuget sign <package folder> -Timestamper http://timestamp.digicert.com -outputdirectory <output folder> -Certificate Fingerprint <SHA1 Thumbprint>
 -HashAlgorithm SHA256 -Verbosity detailed -Overwrite
```

The timestamping URL, certificate fingerprint and Hashing Algorithm are auto generated in the README after running the SIGN+ Installer.

Command sample:

```
nuget sign HelloWorld.1.3.0.17\* -Timestamper http://timestamp.digicert.com -outputdirectory ..\am-HelloWorld.1.3.0.17 -CertificateFingerprint
 4610fdca3ed589qde10235ce687ea1g02043e439 -HashAlgorithm SHA256 -Verbosity detailed -Overwrite
```

Esptool

Esptool is a native sign tool used for Espressif chips, facilitating firmware signing and flashing onto devices.

Sign Secure Boot V2 images with Esptool from Espressif

Esptool is a Python-based, open-source, platform-independent utility to communicate with the ROM bootloader in Espressif chips.

Espressif with AppViewX PKCS11 Provider only supports:

- RSA 3072 bit keys.
- ECDSA 256 bit keys.

Prerequisites

1. Run the AppViewX SIGN+ Installer to install the prerequisites required to use the AppViewX PKCS#11 Provider with Esptool.
2. Python 3.7 or newer Installed.

Install Esptool

To install Esptool, run the following command from command line:

```
pip install esptool[hsm]
```

For additional information refer Esptool Installation and Configuration

Create configuration file

Sample HSM Configuration File:

```
[hsm_config]

pkcs11_lib =<path to AppViewX PKCS11 library>

credentials =NONE

slot =1

label =<keypair-alias>
```

The HSM Configuration file is autogenerated as part of running the SIGN+ Installer.

Sign Command

```
espsecure.py sign_data --version 2 --hsm --hsm-config hsm-config.ini --output v2-rsa-pss-hello_world.bin hello_world.bin
```

Verify Command

```
espsecure.py verify_signature --version 2 --keyfile <public-key-file-of-keypair> <image-file-to-verify>
```

The steps required to generate the public key file for verification are auto generated in the README as part of running the SIGN+ Installer.

XMLSecTool

A command-line tool for signing and verifying XML documents using digital signatures.

Sign XML files with Xmlsectool

Prerequisites

- Run the AppViewX SIGN+ Installer to install the prerequisites required to use the AppViewX PKCS11 Provider with Xmlsectool.
- Download xmlsectool.
- Java_home path set.
- XML file that needs signing.

Note: This file natively runs on Linux and Mac OS. However, Windows requires transferring software (eg.such as Putty) to connect with a Linux terminal to run the .sh files.

Tool Usage and Steps

1. Download xmlsectool zip file.
2. Unzip the downloaded file.
3. Sign in to your console.
4. Copy the XML document to your Linux location.
5. Set up the PKCS11 configuration file.
6. Use the sign XML command.
7. Use the verify XML command.

XML commands

Sign XML file

Command:

```
./xmlsectool.sh --sign --pkcs11Config <path to PKCS11 config file> --keyAlias  <keypair alias> --keyPassword NONE --inFile  <name of xml file to be signed>
 --outFile  <name of xml file after signing>
```

The path to PKCS11 Config File and Keypair Alias are auto generated in the README after running the SIGN+ Installer.

Output sample:

```
./xmlsectool.sh --sign --pkcs11Config pkcs11properties.cfg --keyAlias  TestCert --keyPassword NONE --inFile  UnsignedFileName.xml --outFile
 SignedFileName.xml
INFO XMLSecTool - Reading XML document from file UnsignedFileName.xml
INFO XMLSecTool - XML document parsed and is well-formed.
INFO XMLSecTool - XML document successfully signed
INFO XMLSecTool - XML document written to file /Users/Name/SignedFileName.xml
```

Verify signed XML file

Command:

```
./xmlsectool.sh --verifySignature --pkcs11Config <path to PKCS11 config file> --keyAlias  <keypair alias> --keyPassword  NONE --inFile  <name of xml file after
 signing>
```

The path to PKCS11 Config File and Keypair Alias are auto generated in the README after running the SIGN+ Installer.

Output sample:

```
./xmlsectool.sh --verifySignature --pkcs11Config pkcs11properties.cfg --keyAlias  KeypairAliasExample  --keyPassword  NONE --inFile  SignedFileName.xml

INFO XMLSecTool - Reading XML document from file 'SignedFileName.xml'

INFO XMLSecTool - XML document parsed and is well-formed.

INFO XMLSecTool - XML document signature verified.
```

Mage

Mage (Manifest Generation and Editing Tool) is a command-line tool used for signing manifests and creating deployment manifests for ClickOnce applications. The mage.exe tool is included with the .NET Framework SDK and can also be accessed via the Visual Studio Command Prompt.

Mage's Signing Capabilities

Mage can be used to sign:

- .manifest
- .application

Download Mage

1. Download **mage.exe** by installing Windows SDK. Mage is automatically installed with Visual Studio.
2. Mage file location:

```
C:\Program Files (x86)\Microsoft SDKs\Windows\v10.0A\bin\NETFX 4.8 Tools
```

Sign Manifest Files with Mage using AppViewX CSP

Prerequisites:

1. Execute the AppViewX SIGN+ Installer to set up the prerequisites for using the AppViewX CSP.
2. Installed **mage.exe**.

> **Note:** When you sign a manifest file with mage, AppViewX SIGN+ uses two signature units. This is due to the creation of two hash signatures for different types of content within the manifest.
> • The initial hash signature is created for the manifest file itself.
> • The second hash signature is created for the files referenced within the manifest.
> This dual-signing process ensures the integrity of both the manifest file and its referenced files, protecting against any tampering attempts.

Sign Command

```
mage.exe -Sign <input_file_path> -TimeStampUri <timestamp_url> -CertHash <SHA1 Thumbprint> -Algorithm <digest algorithm>
```

- **-TimeStampUri <timestamp_url>:** Timestamping URL
- **-CertHash <SHA1 Thumbprint>:** SHA1 Thumbprint of Signing Certificate
- **-Algorithm <digest algorithm>:** Hashing Algorithm.

The <timestamp_url>, <SHA1 Thumbprint> and <digest algorithm> are auto generated in the README after running the SIGN+ Installer.

Verify Command

```
mage -verify <file_name>
```

Set-AuthenticodeSignature

The Set-AuthenticodeSignature cmdlet applies an Authenticode signature to any file that supports Subject Interface Package (SIP). This cmdlet is available only on the Windows platform.

Sign Authenticode Files with Set-AuthenticodeSignature using AppViewX CSP

Prerequisites:

1. Execute the AppViewX SIGN+ Installer to install the necessary prerequisites for using the AppViewX CSP.
2. Installed Powershell.

Sign Command:

```
$SigningCertificate = (Get-ChildItem -Path Cert:\\CurrentUser\\My\\<SHA1 Thumbprint>)

Set-AuthenticodeSignature -FilePath <input file> -Certificate $SigningCertificate -TimestampServer <timestamp_url> -HashAlgorithm <digest algorithm>
```

- **-CertHash <SHA1 Thumbprint>:** SHA1 Thumbprint of Signing Certificate
- **-TimeStampUri <timestamp_url>:** Timestamping URL
- **-Algorithm <digest algorithm>:** Hashing Algorithm.

The <SHA1 Thumbprint>, <timestamp_url> and <digest algorithm> are auto generated in the README after running the SIGN+ Installer.

Copy the above command generated in the README File and use it in your PowerShell script or CI/CD pipeline jobs as needed.

Sample Output:

Verify Command

```
(Get-AuthenticodeSignature <signed file path>).Status -eq 'Valid'
```

Sample Output

Cosign

CoSign is an open-source command-line tool designed to enhance container image security by simplifying the signing and verification process. It uses digital signatures, allowing a container image to be signed with a private key, which can then be verified by the recipient using the corresponding public key. This method helps protect against man-in-the-middle (MITM) attacks and ensures that images remain unaltered during distribution.

As part of the Sigstore project, CoSign command supports container signing, verification, and storage in an Open Container Initiative (OCI) registry, facilitating a signatures-invisible infrastructure for data center operations. The goal of the Sigstore project is to enable developers to securely sign software artifacts,

including release files, binaries, container images, bill of materials manifests and more. For more details, see Cosign - Sigstore.

Download Cosign

1. Download and install CoSign version 1.3 or newer, ensuring it has PKCS#11 key support enabled.
2. Download Link: Cosign Download Link.

Prerequisites

1. Execute the AppViewX SIGN+ installer to set up the prerequisites for using the AppViewX PKCS#11 provider with CoSign.
2. Ensure `pkcs11-tool` is pre-installed.
3. Use a version of CoSign that supports the pre-installed `pkcs11-tool`.

> **Note:** Access to a container registry is required for CoSign to function. ttl.sh provides free, short-lived (hours) anonymous container image hosting. Use the following commands to create a short-lived temporary container.

```
IMAGE_NAME=$(uuidgen)

IMAGE=ttl.sh/$IMAGE_NAME:1h

<path to cosign> copy alpine $IMAGE
```

The following examples use a sample container image created from the above command.

Signing and Verifying Container Images with CoSign

1. Verify that the `pkcs11` token can be loaded in CoSign.

```
<path to cosign executable> pkcs11-tool list-tokens --module-path <path to PKCS11.so>
```

Sample Command:

```
<path to cosign> pkcs11-tool list-tokens --module-path "/home/admin/AppViewX Sign+/AVXPKCS11.so"
```

Sample Output:

```
Listing tokens of PKCS11 module '/home/admin/AppViewX Sign+/AVXPKCS11.so'

Token in slot 0

 Label: AppViewX PKCS11

 Manufacturer: AppViewX Inc.
```

```
Model: V2
S/N: 1E7218780068003
```

2. Fetch and list all key URIs from the installed SIGN+ package.

Sample Command:

```
<path to cosign> pkcs11-tool list-keys-uris --module-path "/home/admin/AppViewX Sign+/AVXPKCS11.so" --slot-id 0 --pin 12345678
```

Command Output:

```
Listing URIs of keys in slot '0' of PKCS11 module '/home/admin/AppViewX Sign+/AVXPKCS11.so'
Object 0
 Label: AppViewX Inc Test's AppViewX Intermediate CA
 ID: 323030
 URI:
 pkcs11:token=AppViewX%20PKCS11;slot-id=0;id=%32%30%30;object=AppViewX%20Inc%20Test's%20AppViewX%20Intermediate%20CA?module-pat
h=/home/admin/AppViewX%20Sign+/AVXPKCS11.so&pin-value=12345678
Object 1
 Label: AppViewX Inc Prod's AppViewX Intermediate CA
 ID: 323033
 URI:
 pkcs11:token=AppViewX%20PKCS11;slot-id=0;id=%32%30%33;object=AppViewX%20Inc%20Prod's%20AppViewX%20Intermediate%20CA?module-pat
h=/home/admin/AppViewX%20Sign+/AVXPKCS11.so&pin-value=12345678
```

3. Sign a container image.

```
<path to cosign executable> sign --key <URI> $IMAGE
```

Sample Command:

```
<path to cosign> sign --key
 "pkcs11:token=AppViewX%20PKCS11;slot-id=0;id=%32%30%33;object=AppViewX%20Inc%20Prod's%20AppViewX%20Intermediate%20CA?module-pat
h=/home/admin/AppViewX%20Sign%2B/AVXPKCS11.so&pin-value=12345678" $IMAGE
```

Command Output:

```
Pushing signature to: ttl.sh/4829a8e9-605e-483d-b137-16a003b91d64
```

4. Verify the signed container image:

```
<path to cosign executable> verify --key <URI> $IMAGE
```

Sample Command:

```
<path to cosign> verify --key
 "pkcs11:token=AppViewX%20PKCS11;slot-id=0;id=%32%30%33;object=AppViewX%20Inc%20Prod's%20AppViewX%20Intermediate%20CA?module-pat
h=/home/admin/AppViewX%20Sign%2B/AVXPKCS11.so&pin-value=12345678" $IMAGE
```

Command Output:

```
Verification for ttl.sh/4829a8e9-605e-483d-b137-16a003b91d64:1h --
The following checks were performed on each of these signatures:
  - The cosign claims were validated
  - The signatures were verified against the specified public key

[{"critical":{"identity":{"docker-reference":"ttl.sh/4829a8e9-605e-483d-b137-16a003b91d64"},"image":{"docker-manifest-digest":"sha256:0a4eaa0eecf5f8c050
e5bba433f58c052be7587ee8af3e8b3910ef9ab5fbe9f5"},"type":"cosign container image signature"},"optional":{"Subject":""}}]
```

Signing and Verifying Blob Files with CoSign

1. Verify that the PKCS#11 token can be loaded in CoSign.

```
<path to cosign executable> pkcs11-tool list-tokens --module-path <path to PKCS11.so>
```

Sample Command:

```
<path to cosign> pkcs11-tool list-tokens --module-path "/home/admin/AppViewX Sign+/AVXPKCS11.so"
```

Sample Output:

```
Listing tokens of PKCS11 module '/home/admin/AppViewX Sign+/AVXPKCS11.so'
Token in slot 0
 Label: AppViewX PKCS11
 Manufacturer: AppViewX Inc.
 Model: V2
 S/N: 1E7218780068003
```

2. Fetch and list all key URIs from the installed SIGN+ package.

Sample Command:

```
<path to cosign> pkcs11-tool list-keys-uris --module-path "/home/admin/AppViewX Sign+/AVXPKCS11.so" --slot-id 0 --pin 12345678
```

Command Output:

```
Listing URIs of keys in slot '0' of PKCS11 module '/home/admin/AppViewX Sign+/AVXPKCS11.so'
Object 0
 Label: AppViewX Inc Test's AppViewX Intermediate CA
 ID: 323030
```

```
URI:
pkcs11:token=AppViewX%20PKCS11;slot-id=0;id=%32%30%30;object=AppViewX%20Inc%20Test's%20AppViewX%20Intermediate%20CA?module-path=/home/admin/AppViewX%20Sign+/AVXPKCS11.so&pin-value=12345678
Object 1
Label: AppViewX Inc Prod's AppViewX Intermediate CA
ID: 323033
URI:
pkcs11:token=AppViewX%20PKCS11;slot-id=0;id=%32%30%33;object=AppViewX%20Inc%20Prod's%20AppViewX%20Intermediate%20CA?module-path=/home/admin/AppViewX%20Sign+/AVXPKCS11.so&pin-value=12345678
```

3. Sign a blob file.

```
<path to cosign executable> sign-blob --key <URI> <path_to_blob_file>
```

Sample Command:

```
<path to cosign> sign-blob --key
"pkcs11:token=AppViewX%20PKCS11;slot-id=0;id=%32%30%30;object=AppViewX%20Inc%20Test's%20AppViewX%20Intermediate%20CA?module-path=/home/admin/AppViewX%20Sign%2B/AVXPKCS11.so&pin-value=12345678" README.md
```

Command Output:

```
Using payload from: README.md
RRAvpvZeSXg7lGbCsfL4UAx9u5QHM73t+SaN3BDZ5vvuLKdjWNJRO5PESuuikUBrO9tla5KG7KX6Kk4n2UGG8vEi+CvxJb5iHnO1cju7cHjzpEZWNCg7kRvi
H66uQBGU7LIKiJyAhpz8/slQ2emOuP+2j3+GX7ArJj1/Xy3sLSTn9FRgnl8KtGE8f0K696rRRecFvcOfdLIB4q7w2U2XY7+8x57nhfysRGAd2+GxNwFj+uZst+Kt
5W8JfFbl2lQy0fVnmHNoIyUzdfdRD7zLkLWdKWUSxdZcVSHSHp3QgN+e4fXYWroGVmD8EZWrtiiP8wrkk+hgfa02t/scYROIT0weuFe2abXx114/UsjaxRo10L
xDwsvaH+i6PR4BL02A8FC3oFYWV3Cu7Q/v9zT/0AFh6egONlfYfW98rHhrtuQmoCGvBj9NwcUPwxa3kSC5cAYYByY/17eijiGR2QFt8MBoshluD1ZGsQuOpK
YmkNQjuNNkEEir04KCXoxT/h7+
```

4. Verify a blob File:

```
<path to cosign executable> verify-blob --key <URI> --signature <Base64_SignatureString> <path_to_blob_file>
```

Sample Command:

```
<path to cosign> verify-blob --signature
"RRAvpvZeSXg7lGbCsfL4UAx9u5QHM73t+SaN3BDZ5vvuLKdjWNJRO5PESuuikUBrO9tla5KG7KX6Kk4n2UGG8vEi+CvxJb5iHnO1cju7cHjzpEZWNCg7kRvi
H66uQBGU7LIKiJyAhpz8/slQ2emOuP+2j3+GX7ArJj1/Xy3sLSTn9FRgnl8KtGE8f0K696rRRecFvcOfdLIB4q7w2U2XY7+8x57nhfysRGAd2+GxNwFj+uZst+K
t5W8JfFbl2lQy0fVnmHNoIyUzdfdRD7zLkLWdKWUSxdZcVSHSHp3QgN+e4fXYWroGVmD8EZWrtiiP8wrkk+hgfa02t/scYROIT0weuFe2abXx114/UsjaxRo10L
xDwsvaH+i6PR4BL02A8FC3oFYWV3Cu7Q/v9zT/0AFh6egONlfYfW98rHhrtuQmoCGvBj9NwcUPwxa3kSC5cAYYByY/17eijiGR2QFt8MBoshluD1ZGsQuOpK
YmkNQjuNNkEEir04KCXoxT/h7+" --key
"pkcs11:token=AppViewX%20PKCS11;slot-id=0;id=%32%30%30;object=AppViewX%20Inc%20Test's%20AppViewX%20Intermediate%20CA?module-path=/home/admin/AppViewX%20Sign%2B/AVXPKCS11.so&pin-value=12345678" README.md
```

Command Output:

```
Verified OK
```

Signing and Verifying SBOM Files with CoSign

1. Verify that the PKCS#11 token can be loaded in CoSign.

```
<path to cosign executable> pkcs11-tool list-tokens --module-path <path to PKCS11.so>
```

Sample Command:

```
<path to cosign> pkcs11-tool list-tokens --module-path "/home/admin/AppViewX Sign+/AVXPKCS11.so"
```

Sample Output:

```
Listing tokens of PKCS11 module '/home/admin/AppViewX Sign+/AVXPKCS11.so'
Token in slot 0
 Label: AppViewX PKCS11
 Manufacturer: AppViewX Inc.
 Model: V2
 S/N: 1E7218780068003
```

2. Fetch and list all key URIs from the installed SIGN+ package.

Sample Command:

```
<path to cosign> pkcs11-tool list-keys-uris --module-path "/home/admin/AppViewX Sign+/AVXPKCS11.so" --slot-id 0 --pin 12345678
```

Command Output:

```
Listing URIs of keys in slot '0' of PKCS11 module '/home/admin/AppViewX Sign+/AVXPKCS11.so'
Object 0
 Label: AppViewX Inc Test's AppViewX Intermediate CA
 ID: 323030
 URI:
 pkcs11:token=AppViewX%20PKCS11;slot-id=0;id=%32%30%30;object=AppViewX%20Inc%20Test's%20AppViewX%20Intermediate%20CA?module-pat
h=/home/admin/AppViewX%20Sign+/AVXPKCS11.so&pin-value=12345678
Object 1
 Label: AppViewX Inc Prod's AppViewX Intermediate CA
 ID: 323033
 URI:
 pkcs11:token=AppViewX%20PKCS11;slot-id=0;id=%32%30%33;object=AppViewX%20Inc%20Prod's%20AppViewX%20Intermediate%20CA?module-pat
h=/home/admin/AppViewX%20Sign+/AVXPKCS11.so&pin-value=12345678
```

3. Generate an SBOM file for a container image.

Sample Command:

```
syft $IMAGE -o spdx > sbom_output_file.spdx
```

Command Output:

```
✓ Parsed image                    sha256:324bc02ae1231fd9255658c128086395d3fa0aedd5a41ab6b034fd649d1a9260
✓ Cataloged contents                    eddacbc7e24bf8799a4ed3cdcfa50d4b88a323695ad80f317b6629883b2c2a78
  ├── ✓ Packages                  [14 packages]
  ├── ✓ File digests              [77 files]
  ├── ✓ File metadata             [77 locations]
  └── ✓ Executables               [17 executables]
```

4. Attach the SBOM to the container image.

```
<path to cosign executable> attach sbom --sbom <sbom_file.spdx> $IMAGE  # get the digest from the output
```

Sample Command:

```
<path to cosign> attach sbom --sbom sbom_output_file.spdx  $IMAGE
```

Command Output:

```
WARNING: Attaching SBOMs this way does not sign them. If you want to sign them, use 'cosign attest -predicate sbom_output_file.spdx -key <key path>' or
 'cosign sign -key <key path> <sbom image>'.
Uploading SBOM file for [ttl.sh/4829a8e9-605e-483d-b137-16a003b91d64:1h] to
 [ttl.sh/4829a8e9-605e-483d-b137-16a003b91d64:sha256-0a4eaa0eecf5f8c050e5bba433f58c052be7587ee8af3e8b3910ef9ab5fbe9f5.sbom] with
 mediaType [text/spdx].
```

5. Sign the SBOM.

```
<path to cosign executable> sign --key <URI> <output of attach SBOM command>
```

Sample Command:

```
<path to cosign> sign --key
 "pkcs11:token=AppViewX%20PKCS11;slot-id=0;id=%32%30%33;object=AppViewX%20Inc%20Prod's%20AppViewX%20Intermediate%20CA?module-pat
h=/home/admin/AppViewX%20Sign%2B/AVXPKCS11.so&pin-value=12345678"
 ttl.sh/4829a8e9-605e-483d-b137-16a003b91d64:sha256-0a4eaa0eecf5f8c050e5bba433f58c052be7587ee8af3e8b3910ef9ab5fbe9f5.sbom
```

Command Output:

```
Pushing signature to: ttl.sh/4829a8e9-605e-483d-b137-16a003b91d64
```

6. Verify the SBOM.

```
<path to cosign executable> verify --key <URI> <SBOM>
```

Sample Command:

```
<path to cosign> verify --key
 "pkcs11:token=AppViewX%20PKCS11;slot-id=0;id=%32%30%33;object=AppViewX%20Inc%20Prod's%20AppViewX%20Intermediate%20CA?module-pat
h=/home/admin/AppViewX%20Sign%2B/AVXPKCS11.so&pin-value=12345678"
 ttl.sh/4829a8e9-605e-483d-b137-16a003b91d64:sha256-0a4eaa0eecf5f8c050e5bba433f58c052be7587ee8af3e8b3910ef9ab5fbe9f5.sbom
```

Command Output:

```
Verification for ttl.sh/4829a8e9-605e-483d-b137-16a003b91d64:sha256-0a4eaa0eecf5f8c050e5bba433f58c052be7587ee8af3e8b3910ef9ab5fbe9f5.sbom --
The following checks were performed on each of these signatures:
  - The cosign claims were validated
  - The signatures were verified against the specified public key

[{"critical":{"identity":{"docker-reference":"ttl.sh/4829a8e9-605e-483d-b137-16a003b91d64"},"image":{"docker-manifest-digest":"sha256:f4f7b7532e3830962a
95ff42e226e9b354cb4d5d6922577a3bbc4b2291a9bb1e"},"type":"cosign container image signature"},"optional":{"Subject":""}}]
```

OpenSSL

OpenSSL is a widely used, open-source toolkit that provides a comprehensive set of cryptographic functions, including the ability to generate, manage, and verify digital signatures. As a native signing tool, OpenSSL allows users to sign data, such as files, using private keys, ensuring integrity and authenticity.

Configure OpenSSL for signing with PKCS#11 Engine

OpenSSL is a versatile open-source cryptography library that provides a set of tools and libraries for secure communications and digital signatures. OpenSSL can be integrated with a PKCS#11 Provider using OpenSSL PKCS#11 Engine to sign and verify files.

Prerequisites to sign files using OpenSSL

- OpenSSL 3.x.x Installed.
- Setup OpenSSL PKCS#11 engine.

Install OpenSSL

Linux

To install OpenSSL, OpenSSL PKCS11 engine and P11tool, execute the following command:

- **Ubuntu**

```
sudo apt install -y openssl libengine-pkcs11-openssl gnutls-bin xxd
```

- **RHEL**

```
sudo dnf install openssl openssl-pkcs11 gnutls vim-common
```

Setup and configure PKCS11 library

A configuration file is required for OpenSSL PKCS#11 engine to use AppViewX PKCS#11 library. This file is required in related sign commands.

```
openssl_conf = openssl_init

[openssl_init]

engines = engine_section

[engine_section]

pkcs11 = pkcs11_section

[pkcs11_section]

#Path to the OpenSSL PKCS11 Engine

dynamic_path = "<Path to libpkcs11.so>"

MODULE_PATH = <Path to AVXPKCS11.so>
```

Note:

- The above openssl.conf file is auto-generated after running the SIGN+_Installer executable. It consists of the required openssl configuration details and the **<Path to AVXPKCS11.so>** is dynamically generated after installation. This file is required for using openssl with AppViewX PKCS#11 provider to sign and verify files.
- The libpkcs11.so path depends on the Linux distribution and OpenSSL version and has to be manually entered in the generated **openssl.conf** file.

Note: By default, the README File and the openssl.conf file is generated with the default location of libpkcs11.so in Ubuntu 20.04, 22.04, 24.04 and OpenSSL Version 3. Modify the path based on OS or any custom location based on requirement.

- **Ubuntu 20.04, 22.04, 24.04:** /usr/lib/x86_64-linux-gnu/engines-3/libpkcs11.so
- **RHEL 9:** /usr/lib64/engines-3/libpkcs11.so

Sign software artifacts with OpenSSL using AppViewX PKCS11 library and PKCS#11 Engine

Prerequisites

1. Configure Openssl to sign using PKCS11 Engine.
2. Run the AppViewX SIGN+ Installer to install the prerequisites required to sign using AppViewX PKCS#11 Provider.
3. Modify the configurations in the generated openssl.conf file based on requirements.

Sign with OpenSSL dgst

Sample Command

```
OPENSSL_CONF=<path_to_openssl_conf_file> openssl dgst -engine pkcs11 -keyform engine -sign

"pkcs11:object=<key_alias_name>;type=private" <digest_algorithm> -out <signed output file>

<file_to_sign>
```

The **<path_to_openssl_conf_file>**, **<key_alias_name>** and **<digest algorithm> parameters** are auto generated based on the signing policy configurations in the README after running the SIGN+ Installer.

Verify with OpenSSL dgst

Sample Command

1. Extract the public key from the signing certificate

   ```
   openssl x509 -inform der -in <certificate_path> -pubkey -noout > <output_file_path.pem>
   ```

   The **<certificate_path>** parameter is auto generated in the README after running the SIGN+ Installer.
2. Verify the signed file using the above extracted public key openssl

   ```
   openssl dgst -verify <path to public key> -signature <signed output file> <digest algorithm>

   <signed_file_to_verify>
   ```

   The **<digest algorithm>** parameter is auto generated in the README after running the SIGN+ Installer.

Sign Authenticode files with Osslsigncode using AppViewX PKCS11 library and PKCS#11 Engine

Osslsigncode is a signing tool based on OpenSSL and cURL used to sign, timestamp and verify Authenticode on Linux. It can be integrated with a PKCS#11 Provider using OpenSSL PKCS#11 Engine to sign and verify files.

Use Osslsigncode to sign and timestamp Authenticode files such as:

- .arx
- .cbx
- .crx
- .cpl
- .dbx
- .deploy
- .dll
- .drx
- .exe
- .msi
- .msm
- .msp
- .ocx
- .sys

Prerequisites

- Configure Openssl to sign using PKCS#11 Engine
- Run the AppViewX SIGN+ Installer to install the prerequisites required to sign using AppViewX PKCS#11 Provider.

Install Osslsigncode

```
apt install osslsigncode
```

Sign with Osslsigncode

```
osslsigncode sign -pkcs11engine <path_to_libpkcs11.so> -pkcs11module <path_to_AVXPKCS11.so>

-certs <path_to_certificate> -key 'pkcs11:object=<keypair alias>;type=private' -in <file_to

be_signed> -out <output_signed_file> -h <digest algorithm> -t <timestamp_url>
```

- **<path_to_libpkcs11.so>:** Path to openssl engine libpkcs11.so file. By default this is populated with the path of the libpkcs11.so file in Ubuntu 20.04, 22.04, 24.04 and OpenSSL Engine 3.
- **<path_to_AVXPKCS11.so>:** Path to AVXPKCS11.so file.
- **<path_to_certificate>:** Path to the signing certificate.
- **<keypair alias>:** Alias Name of the Signing Certificate.
- **<digest_algorithm>:** Specifies the hashing algorithm.
- **<timestamp_url>:** Specifies the timestamping URL.
- **<file_to be_signed>:** Path to the file to be signed.
- **<output_signed_file>:** Path of signed file.

The **<path_to_libpkcs11.so>**, **<path_to_AVXPKCS11.so>**, **<path_to_certificate>**, **<keypair alias>**, **<digest algorithm>** and **<timestamp_url>** parameters are auto generated based on the signing policy configurations in the README after running the SIGN+ Installer.

Note: Osslsigncode throws an error during signing if the output signed file already exists.

Troubleshooting Guide for SIGN+ Native Tools Integration

This guide provides solutions for common issues encountered when integrating SIGN+ with native tools. It covers troubleshooting techniques for installation errors, configuration issues to ensure smooth and efficient use of SIGN+.

Log Files Path

SIGN+_Installer logs

AppViewX SIGN+ creates log files on Linux, macOS, and Windows, organized by the current date. This allows users to track file-signing activities and review any related error traces.

Windows

The SIGN+ installer logs for Windows are stored in the temporary folder. The log file can be found at:

```
C:\Users\<user>\AppData\Local\Temp\SIGN+Installer_Logs.log
```

Alternatively, users can enter **%tmp%** in File Explorer to quickly access the temporary folder and find the **SIGN+Installer_Logs.log** file.

Linux and MacOS

On Linux and macOS, the SIGN+ installer log files can be found at:

```
/tmp/SIGN+Installer_Logs.log
```

AppViewX CSP Logs

Windows

The SIGN+ CSP logs for Windows are stored in the **tmp** folder. The installer log can be found at the following path:

```
C:\Users\<user>\AppData\Local\Temp\AppViewX_CSP_Logs_<Day>.log
```

Alternatively, users can type **%tmp%** in File Explorer to quickly locate the **AppViewX_CSP_Logs_<Day>.log** file.

```
Example: C:\Users\<user>\AppData\Local\Temp\AppViewX_CSP_Logs_Monday.log
```

A separate log file is created for each day of the week, and the logs are overwritten in the following week.

AppViewX PKCS#11 Logs

Windows

The SIGN+ PKCS11 logs for Windows are stored in the **tmp** folder. The installer log can be found at the following path:

```
C:\Users\<user>\AppData\Local\Temp\AvxPKCS11_<Day>.log
```

Alternatively, users can type **%tmp%** in File Explorer to access the folder containing the **AvxPKCS11_<Day>.log** file.

Example: C:\Users\<user>\AppData\Local\Temp\AvxPKCS11_Monday.log

A separate log file is created for each day of the week, and the logs are overwritten in the following week.

Linux and MacOS

On Linux and macOS, the SIGN+ installer log files can be found at:

/tmp/AvxPKCS11_<Day>.log

Example: /tmp/AvxPKCS11_Monday.log

A separate log file is created for each day of the week, and the logs are overwritten in the following week.

Common Errors and Solutions

Compute Cluster or Cloud Connector Connectivity

Error Message

Error Performing SSL/TLS Handshake

Problem

This error occurs when the machine is unable to establish a connection with a required hostname. This could be due to:

- **For SaaS deployment of SIGN+:** Issues resolving the Cloud Connector (CC) hostname.
- **For On-Prem deployment of SIGN+:** Issues resolving any other hostname (e.g., any service).
- Check the **Firewall** rules and ensure that connections to the target URL (**Compute Cluster/Cloud Connector/On-Prem**) are allowed.

Solution

Ensure the Compute Cluster or Cloud Connector is reachable from the machine and is able to resolve the Cloud Connector Hostname.

Windows:

1. Open the ***C:\Windows\System32\drivers\etc\hosts*** file as Administrator.
2. Add the following entry: **<IP Address> <Hostname>**

- Replace **<IP Address>** with the IP address of the server or the target machine.
- Replace **<Hostname>** with the required hostname (e.g., the Cloud Connector, Compute Cluster, or any other service hostname).

3. Save the file.
4. Retrigger the signing command from the signing tool.

Linux and MacOS

1. Open ***/etc/hosts*** file as Administrator.
2. Add the following entry: **<IP Address> <Hostname>**
   - Replace **<IP Address>** with the IP address of the server or the target machine.
   - Replace **<Hostname>** with the required hostname (e.g., the Cloud Connector, Compute Cluster, or any other service hostname).
3. Save the file.
4. Retrigger the signing command from the signing tool.

Additional Notes

- Ensure the hostname is configured correctly and the server is reachable from the machine.
- This solution applies to any hostname resolution issues, including the Cloud Connector, Compute Cluster, or any other service involved in the operation.

Cloud Connector Version Upgrade

Error Message

```
Message : Resource not found, reason - Invalid apiid
```

Problem

This error occurs due to one of the following reasons:

- A mismatch between the Cloud Connector version and the expected AppViewX release version.
- The CodeSigning pod is not up and running in the AppViewX deployment.

Solution

Scenario 1: Cloud Connector Version Mismatch

Ensure the Cloud Connector is upgraded to the latest version compatible with the AppViewX Compute Cluster. Upgrading ensures access to the latest set of APIs.

Scenario 2: CodeSigning Pod Not Running in On-prem deployment

1. Post installation, the admin Kubernetes config is replaced by a read-only user config.
2. The admin config backup can be found at **/home/appviewx/.kube/backup/config.**
3. For administrative activities (scripts or operations other than **get** and **list** Kubernetes API calls), revert to the admin config as follows:
   - Backup the current read-only config: **cp /home/appviewx/.kube/config /home/appviewx/.kube/backup/read-only-config**
   - Replace it with the admin config: **cp /home/appviewx/.kube/backup/config /home/appviewx/.kube/config**
   - Once the required activity is completed, revert back to the read-only user config: **cp /home/appviewx/.kube/backup/read-only-config /home/appviewx/.kube/config**
4. Go to this location: **cd /home/appviewx/appviewx_kubernetes/scripts**
5. Open appviewx.conf and add this line instead of the ENABLED_PLUGINS, **ENABLED_PLUGINS=appviewx_dependencies,avx_platform_gateway,avx_subsystem_codesigning**
6. Execute this command

```
./plugins_install.sh
```

Scenario 3: CodeSigning Pod Not Running in SaaS deployment

If the CodeSigning pod is down, follow these steps to bring it back up:

1. Launch **k9s** and navigate to the deployments view by typing :**deploy** and pressing Enter.
2. Identify deployments with non-critical pods that can be scaled down to free up resources.
   - Select such deployments to edit the YAML configuration, and set **replicas**: **0**
   - Save the changes **(Ctrl + S)** to scale down the pods.
3. Locate the CodeSigning pod's deployment and edit its YAML.
   - Set the **replicas** field to **1** (or more) to attempt to bring the pod back up.
   - Save the changes **(Ctrl + S)** to apply.
4. Monitor the pod status in k9s to confirm that the CodeSigning pod is running.
   - Replicas under the spec section to 0/1

Additional Notes

- Always ensure that critical pods like CodeSigning are prioritized when managing resources.
- Regularly update the Cloud Connector and associated components to ensure compatibility with the AppViewX release.
- Using k9s provides a convenient way to manage pod scaling without requiring **kubectl** commands.

Hash Algorithm Mismatch

Error Message (Hash Algorithm Mismatch):

The input hash is not computed with the hashing algorithm configured in the selected Policy.

Problem

This error occurs when there is a mismatch between the **Configured Hash Function** in the signing policy and the **Hash Function** used in the signing tool commands. When these two configurations are not aligned, the signing operation fails as the input hash does not meet the expected algorithm requirements.

Note: The **hashing algorithm** configured in the **Signing Policy** will be included by **default** in the generated **README**.

Solution

There are two ways to resolve this issue:

1. **Update the Signing Command:**
   - Refer to the **README** file generated after running the SIGN+ Installer.
   - Ensure the signing tool command uses the same hash function configured in the signing policy.
   - After ensuring the command matches the configured policy, retrigger the signing operation.
2. **Modify the Signing Policy:** If the hash function used in the signing tool is preferred, update the signing policy to align with this hash function:
   - Access the signing policy configuration through the SIGN+ administrative interface.
   - Change the **Configured Hash Function** to the desired hash algorithm (e.g., SHA-256, SHA-512).
   - Save the changes and reattempt the signing process.

Unsupported Signing Type

Error Message (Unsupported Signing Type):

Your chosen signing type is not supported by the selected policy.

Problem

This error occurs when the **SIGN+ Package** is downloaded with a **File Based Signing Policy** (which doesn't support Hash Based Signing operations). SIGN+ package and installer are used for **Hash-Based Signing** operations. The mismatch between the signing type specified in the Signing policy and the signing type used by the signing tools causes this issue.

Solution

To resolve this issue, you can choose one of the following approaches:

1. **Use a Compatible SIGN+ Package:**
   • Download the SIGN+ package with policies configured specifically for **Hash Based Signing**.
   • Run the installer included in this package to set up the appropriate environment.
   • After installation, execute the signing commands again.
2. **Update the Signing Policy:**
   • Access the SIGN+ administrative interface.
   • Change the signing type in the **Signing Policy** from **File Based** to **Hash Based**.
   • Save the updated policy and retrigger the signing commands.

Unable to Load Certificates

Error Message (Unable to Load Certificates):

Error Loading Certificates from User AppData Folder. Unknown Directories or Files Found during installation in the SIGN+_Package folder. Please retry installation after moving them to a different location.

Problem

This issue occurs when the **SIGN+_Package** folder contains additional, unknown directories that were not part of the original package. These extraneous directories interfere with the installation process, preventing the application from properly loading the required certificates from the **User AppData** folder.

Solution

To resolve this issue:

1. Inspect the **SIGN+_Package** folder for any additional or unknown directories that were not part of the original download. Move these directories to a different location outside the SIGN+_Package folder.
2. Once the folder contains only the files originally included in the downloaded **SIGN+_package**, rerun the installer and ensure the installation process completes successfully without errors.
3. When rerunning the installer, select **No** for the prompt: **Do you want to update the existing configurations along with the newly downloaded Policies?** to Overwrite existing installation.
4. **Retry the Signing Command:** After the installation is complete, rerun the signing command that was generated in the **README**.

Signtool Errors and Solutions

SignTool: No Private Key Error

Error Message

```
SignTool Error: No private key is available.
```

Problem

This error message appears when the CSP Library or its dependent library files have been deleted.

Solution

Re-run the SIGN+_Installer in the downloaded SIGN+_Package to install and copy the required library files and retrigger the signing command.

Non Existent File referenced

Error Message

```
SignTool Error: An unexpected internal error has occurred.

Error information: "Error: Store IsDiskFile() failed." (-2147024893/0x80070003)
```

Problem

This error message appears when the **/f** parameter in the signtool command points to a non-existent file.

Solution

Re-run the SIGN+_Installer in the SIGN+_Package to install and copy the required files and use the newly generated command in the README.

SignTool Verification error

Error Message

```
Error Message: SignTool Error: WinVerifyTrust returned error: 0x800B010A

A certificate chain could not be built to a trusted root authority.
```

Problem

This error occurs when the file has been signed using a non-trusted certificate (when the customer signs using the appviewx certificate) or the certificate is not imported into the windows certificate store.

Solution

Sign with a trusted Certificate or import the certificate into the windows certificate store and try verifying the file again using the signtool command.

Jarsigner Errors and Solutions

Certificate Chain Not Found Error

Error Message

```
jarsigner: Certificate chain not found for: <Certificate Alias>.  <Certificate Alias> CA must reference a valid KeyStore key entry containing a private key and corresponding public key certificate chain.
```

Problem

This error message appears when the **Certificate Alias** provided is incorrect or the certificate corresponding to the alias has been deleted in the Windows Key Storage. Without a valid alias referencing a certificate entry the signing process cannot proceed.

Solution

To resolve this issue, follow these steps:

1. **Verify the Certificate Alias**: Ensure that the **Certificate Alias** provided in the command matches the one that was generated in the **README**.
2. **Reinstall the Required Certificates:**
   - If the certificate is missing or was deleted, rerun the **SIGN+ Installer** to reinstall the necessary certificates.
   - Ensure the installer completes successfully without errors.
3. **Retry the Command:**
   - After verifying or reinstalling the certificate, reattempt the command with the correct alias.

Signer’s certificate chain is invalid warning when signing and verifying a jar

Error Message

```
Warning:
The signer's certificate chain is invalid. Reason: PKIX path building failed: sun.security.provider.certpath.SunCertPathBuilderException: unable to find valid certification path to requested target
```

Problem

This warning occurs when signing or verifying a **JAR** file using a certificate generated with a private trust. The issue arises because the root and intermediate certificates associated with the private trust have not been imported into the **JDK cacerts KeyStore**. Without these certificates, the Java security framework cannot validate the certificate chain, leading to the invalid chain warning.

Solution

To resolve this warning, you can **either**:

1. **Use a Public Trust**:
   - Use certificates issued by a publicly trusted Certificate Authority (CA). These public CAs are already included in the JDK **cacerts** KeyStore by default, ensuring that the certificate chain can be validated without additional configuration.
2. **Import Certificates into the JDK cacerts KeyStore**:
   - If using a private trust, manually import the **Root CA certificate** and the **Intermediate Issuing CA certificate** into the JDK **cacerts** KeyStore.

Jarsigner: Not a Private Key

Error Message

```
jarsigner: key associated with <Certificate Alias> not a private key
```

Problem

This error occurs when the **AppViewX CSP Library** or its dependent library files are missing or have been deleted. Without these libraries, the system cannot access or associate the private key required for the signing operation.

Solution

To resolve this error, follow these steps:

1. **Reinstall the AppViewX CSP Library**:
   - Locate the **SIGN+_Package** that was previously downloaded.
   - Run the **SIGN+_Installer** included in the package. This installer will reinstall and restore the required library files, including the AppViewX CSP Library and its dependencies.
2. **Retry the Signing Command**: After the installation is complete, rerun the signing command that was generated in the **README**.

Java: ProviderException

Error Message

Jarsigner with PKCS#11 Windows

```
jarsigner error: java.security.ProviderException: Library C:\Windows\System32\AVXPKCS11V1.dll does not exist
```

Jarsigner with PKCS#11 Linux

```
jarsigner error: java.lang.reflect.InvocationTargetException
```

JSign and APKSigner with PKCS#11

```
java.security.ProviderException: Failed to create a SunPKCS11 provider from the configuration <Path to AVXPKCS11V1.cfg>
```

Problem

These error messages occur when the **AppViewX PKCS#11 Library** or its dependent library files are missing, have been deleted, or have been moved from their original installation location. This prevents the signing tools or PKCS11? from accessing the necessary libraries for PKCS#11 operations.

Solution

1. Locate the **SIGN+_Package** that contains the **SIGN+_Installer**.
2. Run the **SIGN+_Installer** to reinstall and restore the AppViewX PKCS#11 Library along with its dependencies. This step will ensure that:
   - The required files (e.g., AVXPKCS11V1.dll for Windows or equivalent libraries for Linux) are copied to their correct locations.
   - The configuration file (AVXPKCS11V1.cfg) is updated and correctly placed.
3. **Verify Library Installation**: After running the installer, ensure that:
   - On Windows, the **AVXPKCS11V1.dll** file is present in the **C:\Windows\System32** directory.
   - On Linux, verify the location of the PKCS#11 library file and ensure the configuration file path matches the installation.
4. **Retry the Signing Command**: Execute the signing command again with the correct configuration and library paths that’s generated in the **README**.

APKSigner Errors and Solutions

Exception in thread "main" java.lang.IllegalAccessException

Error Message

```
Error: Exception in thread "main" java.lang.IllegalAccessException: class com.android.apksigner.ApkSignerTool$ProviderInstallSpec cannot access class sun.security.pkcs11.SunPKCS11 (in module jdk.crypto.cryptoki) because module jdk.crypto.cryptoki does not export sun.security.pkcs11 to unnamed module @72ea2f77
```

Problem

APKSigner is compatible with the following Java versions: Java 8, Java 11, and Java 17.

This issue occurs if a Java version other than the ones listed above is used to run APKSigner.

Solution

To use APKSigner, the system must have either Java 8, Java 11, or Java 17 installed.

JSign Errors and Solutions

net.jsign.timestamp.TimestampingException: Unable to complete the timestamping after 3 attempts

Error Message

```
Error: net.jsign.timestamp.TimestampingException: Unable to complete the timestamping after 3 attempts
```

Problem

This issue occurs when Jsign is unable to connect to the timestamp server. The Jsign library allows a maximum of 3 attempts to reach the server and receive a response.

Additionally, if the user has configured a proxy for their node, the proxy details must be included in the Jsign command.

Solution

If the proxy has not been configured and the timestamp server is unreachable , the user can configure any other timestamp of their choice in the command.

```
java -jar /home/suryanarayan.bhaskar/SIGN+_Files/Jsign_Files/Signing\ tools/jsign-6.0.jar --keystore "/home/suryanarayan.bhaskar/AppViewX Sign+/AVXPKCS11V1.cfg" --storetype PKCS11 --storepass 12345678 --alias "AppViewXCertificate's AppViewX Intermediate CA" --alg "SHA-256" --tsaurl <tsaURL> <input file>
```

If the user has configured the proxy in their node then the command:

```
java -jar /home/suryanarayan.bhaskar/SIGN+_Files/Jsign_Files/Signing\ tools/jsign-6.0.jar --keystore "/home/suryanarayan.bhaskar/AppViewX Sign+/AVXPKCS11V1.cfg" --storetype PKCS11 --storepass 12345678 --alias "AppViewXCertificate's AppViewX Intermediate CA"  -J-Dhttp.proxyHost=<Host> -J-Dhttp.proxyPort=<Port> -J-Dhttp.proxyUser=<Username> -J-Dhttp.proxyPassword=<Password> --alg "SHA-256" --tsaurl <tsaURL> <input file>
```

Where the user has to provide the hostname, port number, username and password for the proxy configured.

ESPTool Errors and Solutions

espsecure.py: command not found

Error Message

```
Error: espsecure.py: command not found
```

Problem

This problem occurs when the system is not able to locate the esp tool even after executing the command:

```
pip install esptool[hsm]
```

Solution

```
sudo find / -name espsecure.py 2>/dev/null
```

This command is used to find the absolute path of the esptool where the path would be **/home/<username>/.local/bin/espsecure.py**

Once the path is located, use this path in the signing command:

```
python3 /home/<username>/.local/bin/espsecure.py sign_data --version 2 --hsm --hsm-config

 "/home/suryanarayan.bhaskar/Downloads/SIGN+_Package_latest/FileBasedPolicy_hsm_config.ini" --output <output_file_path> <input_file_path>
```

Key file has length 4096 bits. Secure boot v2 only supports RSA-3072

Error Message

```
Error : Key file has length 4096 bits. Secure boot v2 only supports RSA-3072.
```

Problem

This problem occurs when the user uses a certificate which does not have a RSA-3072 key.

Solution

ESP Tool works with certificates which have **RSA-3072** or **ECDSA-256** keys. Configure a certificate which has RSA-3072 or ECDSA-256 keys.

XMLSecTool Errors and Solutions

JAVA_HOME Environment Variable Not Set

Error Message

```
Error Message: JAVA_HOME environment variable not set
```

Problem

This Problem occurs when the java path is not set in the environment variable.

Solution

Execute the following commands to include the java path:

- **readlink -f $(which java)**

  This command prints the path of the java location.

  ```
  Example: /usr/lib/jvm/java-21-openjdk-amd64/bin/java
  ```

- Login as root user (**sudo su**) and run the command **sudo vi ~/.bashrc** and include the following lines:
  - export JAVA_HOME=/usr/lib/jvm/java-21-openjdk-amd64

    Enter the path of the java location obtained from the first command here, excluding the **bin/java**.

  - export PATH=$JAVA_HOME/bin:$PATH
- After adding the two lines in the **bashrc** file execute the command **source ~/.bashrc** and then logout from the root user.
- The Java environmental variable will be set by executing the above commands.

Cosign Errors and Solutions

UNAUTHORIZED Error

Error Message

```
Error Message: Error: signing [$IMAGE]: recursively signing: signing digest: POST https://index.docker.io/v2/$IMAGE/blobs/uploads/: UNAUTHORIZED:
authentication required; [map[Action:pull Class: Name:$IMAGE Type:repository] map[Action:push Class: Name:$IMAGE Type:repository]]
```

```
main.go:52: error during command execution: signing [$IMAGE/loginpage]: recursively signing: signing digest: POST
https://index.docker.io/v2/$IMAGE/blobs/uploads/: UNAUTHORIZED: authentication required; [map[Action:pull Class: Name:$IMAGE Type:repository]
map[Action:push Class: Name:$IMAGE Type:repository]]
```

Problem

This issue occurs when the user is not logged into either the Docker registry or the local registry.

Solution

If the image is available in the Docker registry, use the **docker login** command.

If the image is available in the local registry, use the command: **docker login localhost(server_hostname):<port>**.

No such File or Directory Error

Error Message

```
Error: signing [<image_name>]: getting signer: reading key: opening pkcs11 token key: access modulePath: stat /home/<username>/AppViewX Sign /AVXPKCS11.so: no such file or directory
```

```
main.go:52: error during command execution: signing [<image_name>]: getting signer: reading key: opening pkcs11 token key: access modulePath: stat /home/<username>/AppViewX Sign /AVXPKCS11.so: no such file or directory
```

Problem

Cosign does not recognize the + character directly; instead, it recognizes its URL-encoded equivalent %2B.

Solution

This command lists the PKCS11 Key URIs

```
<path to cosign executable> pkcs11-tool list-keys-uris --module-path "/home/username/AppViewX Sign+/AVXPKCS11.so" --slot-id 0 --pin 12345678
```

Replace **+** with **%2B** in the URI while using the URI in the signing commands.

OpenSSL Errors and Solutions

Invalid engine "pkcs11" No engine specified for loading private key / No filename or uri specified for loading private key

Error Message

```
Invalid engine "pkcs11" No engine specified for loading private key / No filename or uri specified for loading private key:
```

Problem

This error occurs when the **libengine-pkcs11-openssl** command is not Executed. This command enables OpenSSL to interact with PKCS#11 modules.

Solution

To enable support for OpenSSL to interact with PKCS#11, run the following command:

```
sudo apt install -y openssl libengine-pkcs11-openssl gnutls-bin xxd
```

Chapter 2: SIGN+ Admin Guide

- Certificate Authority
- Certificate Group
- CA Policy
- Signing Policy
- Sign Logs
- Password Vault
- Configuring Certificate Attributes and Tags
- Configuring Certificate Profiles
- Default User Roles and Permissions
- Expired Certificates
- History of Certificates
- Job Scheduler
- Email Settings
- Self-Service SIGN+: Download Package & Manage Credentials
- Sign Settings

Certificate Authority

- Configuring CA Settings

Configuring CA Settings

- Amazon and Amazon Private CA
- CSC Global CA
- Custom CA
- DigiCert CA
- DigiCert MPKI
- DigiCert One
- EJBCA CA
- Entrust CA
- Entrust MPKI
- GlobalSign MSSL CA

- GlobalSign SSL CA
- GlobalSign Atlas CA
- GoDaddy CA
- Google CA
- HashiCorp Vault CA
- HydrantID CA
- IDnomic CA
- InCommon CA
- Let’s Encrypt CA
- Microsoft Enterprise CA
- Microsoft Standalone CA
- Nexus CA
- Sectigo CA
- SwissSign CA
- Symantec CA
- Trustwave CA

Amazon and Amazon Private CA

Prerequisites

The prerequisites for configuring Amazon CA or Amazon Private CA account in AppViewX are as follows:

- An Amazon account for a user having necessary access for enrolling the certificates and other CLM operations.
- AppViewX server should either have internet access or have a proxy configured in AppViewX general settings. Refer to the section Managing Proxy Settings in the Platform guides.
- Policy JSON for AWS Ec2 Instance Certificate Management.
- Prerequisite for Amazon CA:

```
{
"Version": "2012-10-17",
"Statement": [
{
"Sid": "VisualEditor0",
"Effect": "Allow",
```

```
"Action": [
"ssm:SendCommand",
"ssm:DescribeDocument",
"ec2:DescribeInstances",
"ec2:DescribeRegions",
"s3:ListBucket",
"ssm:CreateDocument",
"ssm:GetCommandInvocation",
"s3:GetObject",
"s3:ListAllMyBuckets",
"ssm:DescribeInstanceInformation",
"ssm:GetDocument",
"s3:DeleteObject",
"s3:GetBucketLocation"
],
"Resource": "*"
}
]
}
```

Policy JSON for Certificate Management in AWS Classic and Application LoadBalancers:

```
{
"Version": "2012-10-17",
"Statement": [
{
"Sid": "VisualEditor0",
"Effect": "Allow",
"Action": [
"iam:GetServerCertificate",
"elasticloadbalancing:DescribeLoadBalancers",
"elasticloadbalancing:ModifyListener",
"elasticloadbalancing:DescribeListeners",
"acm:GetCertificate",
"ec2:DescribeRegions",
"elasticloadbalancing:DescribeTargetHealth",
"acm:ImportCertificate",
"elasticloadbalancing:SetLoadBalancerListenerSSLCertificate",
"iam:UploadServerCertificate"
```

```
],
"Resource": "*"
}
]
}
```

Policy JSON for Certificate Management in AWS Cloudfront:

```
{
"Version": "2012-10-17",
"Statement": [
{
"Sid": "VisualEditor0",
"Effect": "Allow",
"Action": [
"ec2:DescribeRegions",
"cloudfront:ListDistributions",
"cloudfront:UpdateDistribution",
"cloudfront:GetDistributionConfig"
],
"Resource": "*"
}
]
}
```

Policy JSON for IAM Certificate Management:

```
{
"Version": "2012-10-17",
"Statement": [
{
"Sid": "VisualEditor0",
"Effect": "Allow",
"Action": [
"iam:GetServerCertificate",
"iam:UpdateServerCertificate",
"iam:ListServerCertificates",
"ec2:DescribeRegions",
"iam:UploadServerCertificate"
],
"Resource": "*"
```

```
}
]
}
```

Policy JSON for ACM Certificate Management:

```
{
"Version": "2012-10-17",
"Statement": [
{
"Sid": "VisualEditor0",
"Effect": "Allow",
"Action": [
"acm:DescribeCertificate",
"acm:RequestCertificate",
"acm:GetCertificate",
"ec2:DescribeRegions",
"acm:ListCertificates",
"acm:ImportCertificate"
],
"Resource": "*"
}
]
}
```

Prerequisite for Amazon Private CA.

Policies and Permissions required for AWS IAM User:

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Sid": "VisualEditor0",
 "Effect": "Allow",
 "Action": [
 "s3:PutObject",
 "s3:GetObjectAcl",
 "s3:GetObject",
 "s3:PutObjectAcl"
 ],
 "Resource": [
```

```
"arn:aws:s3:::<bucketname>",
"arn:aws:s3:::<bucketname>/*"
]
},
{
"Sid": "VisualEditor1",
"Effect": "Allow",
"Action": [
"acm-pca:GetCertificate",
"ec2:DescribeRegions",
"acm-pca:GetCertificateAuthorityCertificate",
"acm-pca:RevokeCertificate",
"acm:RenewCertificate",
"acm-pca:ListCertificateAuthorities",
"acm-pca:DescribeCertificateAuthorityAuditReport",
"acm-pca:CreateCertificateAuthorityAuditReport",
"s3:ListAllMyBuckets",
"acm:DescribeCertificate",
"acm-pca:IssueCertificate",
"acm:RequestCertificate",
"acm:GetCertificate",
"acm:ListCertificates",
"acm-pca:DescribeCertificateAuthority"
],
"Resource": "*"
}
]
```

AWS Simple Storage Service (S3) Bucket Policy for parsing Audit log:

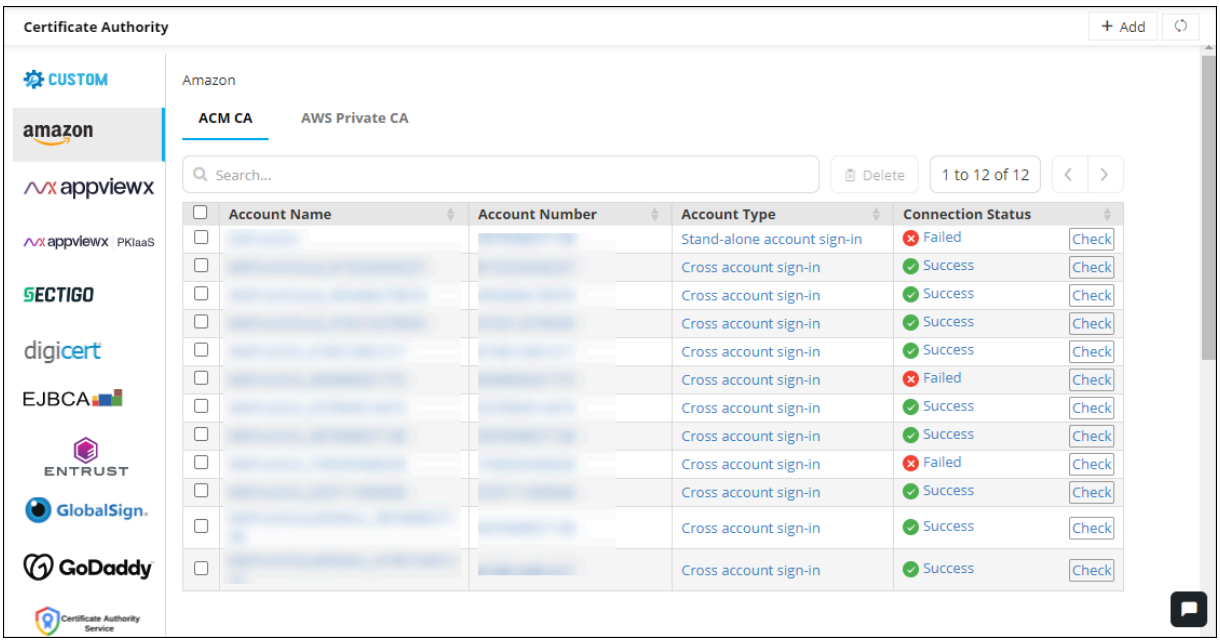
```
{
"Version": "2012-10-17",
"Statement": [
{
"Effect": "Allow",
"Principal": {
"Service": "acm-pca.amazonaws.com"
},
"Action": [
```

```
"s3:PutObject",
"s3:PutObjectAcl",
"s3:GetBucketAcl",
"s3:GetBucketLocation"
],
"Resource": [
"arn:aws:s3:::bucket_name/*",
"arn:aws:s3:::bucket_name"
]
}
]
}
```

Configuring Amazon CA

1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Certificate Authority**.
2. From the displayed CA, Select **Amazon**.

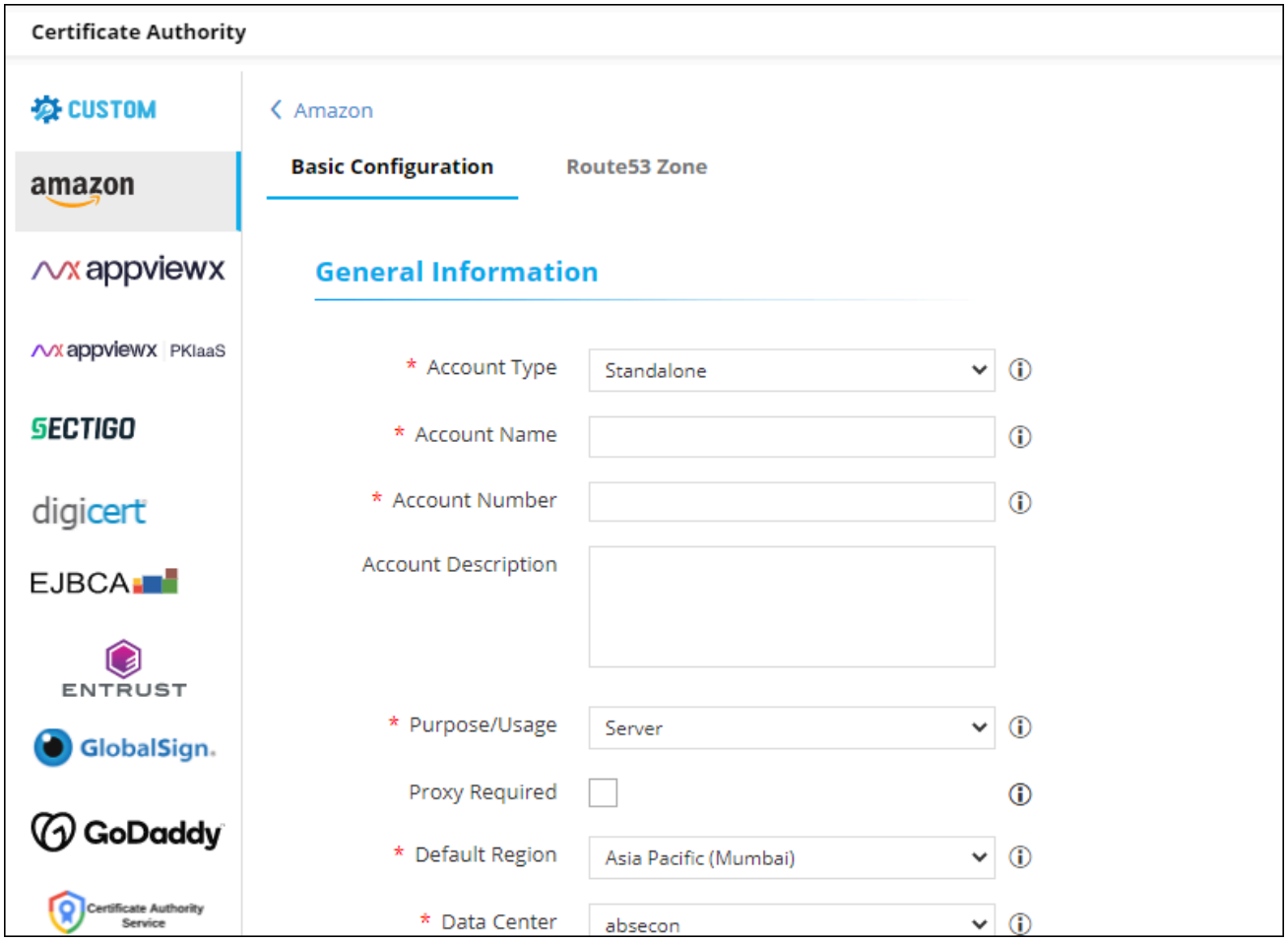
   The **Amazon** home page is displayed.



3. To configure Amazon CA, click **ACM CA** from the home page.
4. Click the **Configure Now** button or **+Add** icon from the middle or top-right of the page respectively.

> **Note:** The **Configure Now** option is displayed if you are configuring a CA for the first time.

The **Amazon** configuration page is displayed.



5. Enter/Select the following details in the **General Information** section:

General Information - Field Description Table

Fields	Description
***Account Type**	From the dropdown list, select one of the following account types: • **Standalone** (Traditional access key- and secret key-based communication) • **Cross or Federated** (Authentication using assume role)
***Account Name**	Unique name for the certificate authority (CA) account represented during certificate enrollment and policy creation
***Account Number**	Valid AWS account number

Fields	Description
Account Description	Additional information related to the CA account being configured
***Purpose/Usage**	Certificate Type for which CLM actions will be enabled. The available options are, • Server • Client.
Proxy Required	Enable this field if the CA communication needs to happen via Proxy. The proxy details configured in general settings will be used for communication.
***Default Region**	Default region for API communication
***Data Center (AppViewX's CA agent)**	Select the data center through which the CA communication needs to happen.
*: *Mandatory fields*	

6. Enter/Select the following **Credentials**-related information:

Credentials - Field Description Table

Fields	Description
Credential type*	From the dropdown list, from the following options, select the credential type: • **Manual Entry**: Manually enter the access and secret key for the customer's AWS account)
Access key*	Enter the access key for the customer's AWS account. **Note:** This field is displayed only when **Credential** type is set to **Manual Entry**.
Secret key*	Enter the secret key for the customer's AWS account.

Fields	Description
	Note: This field is displayed only when **Credential** type is set to **Manual Entry**.
*: *Mandatory fields*	

7. Enter/Select the following details in the **Discover resources** section:

Discover Resources - Field Description Table

Fields	Description
Role ARN for Resource Discovery*	**Note:** This field is displayed only when **Account Type** is **Cross or Federated**. To let the master account assume role for the child account (get temporary privileges to discover resources from the child account), configure the role ARN for resource discovery: a. Click  . b. Enter the following details:

Fields	Description
Role Session name	**Role Session name** is an identifier for the assumed role session. Use the **Role Session name** to uniquely identify a session when the same rule is assumed by different principals or for different reasons.
Duration Seconds	Enter the duration, in seconds, for which the

Fields	Description	
	Fields	Description
		credentials should remain valid. Acceptable durations for IAM user sessions: • **Minimum**: 900 seconds (15 minutes) • **Maximum**: 129,600 seconds (36 hours) **Default**: 3600 seconds (1 hour)
	External Id	**External Id** is a unique identifier that might be required when you assume a role in another account.
	Source Identity	The source identity is specified by the principal that is calling the **AssumeRole** operation.
	Session Tags	**Session Tags** are key-value pairs that you pass when you assume an IAM role or federate a user in AWS STS.

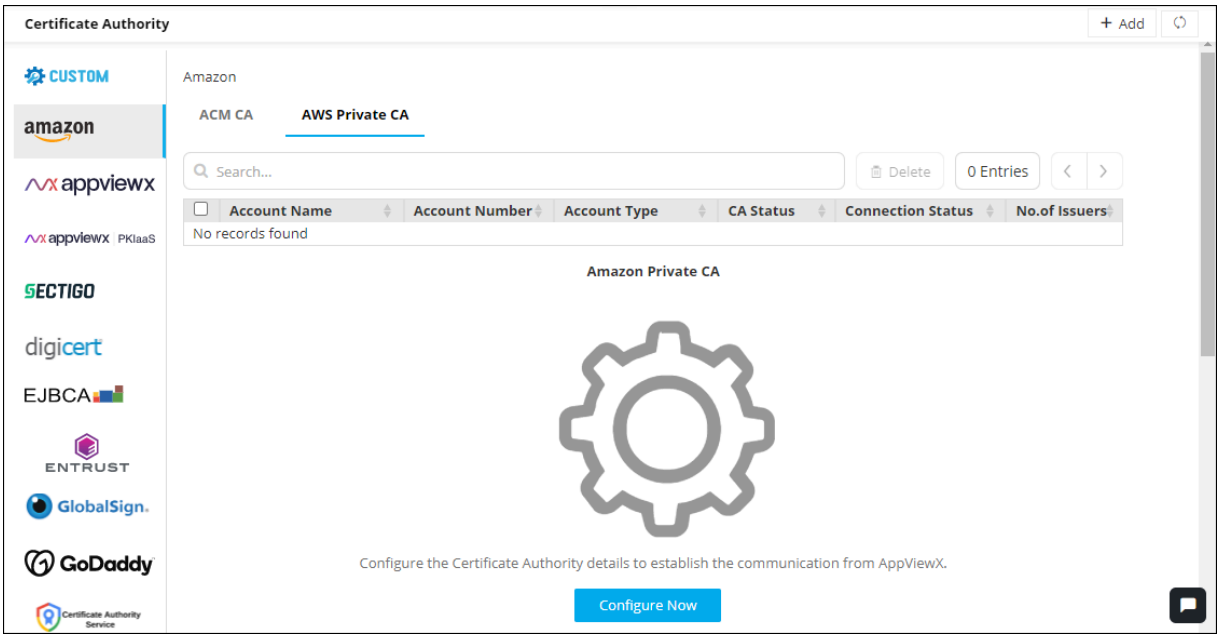
<table>
<tr><th>Fields</th><th>Description</th></tr>
<tr><td></td><td>
<table>
<tr><th>Fields</th><th>Description</th></tr>
<tr><td></td><td>To create a session tag:
i. In the Enter Key field, enter a key for the key-value pair.
ii. In the Enter Value field, enter a value for the key-value pair.
iii. Click Add.
The added key-value pair is shown in the table below the fields.</td></tr>
</table>
</td></tr>
<tr><td>Service Region*</td><td>To select a service region:
a. To fetch the service regions for the account information provided, click Fetch Region.
The retrieved service regions are populated in the Select the Region(s) dropdown list.
b. From the Select the Region(s) dropdown list, select the required service region.</td></tr>
<tr><td>Discover Certificate</td><td>To enable instant certificate discovery at the time of device addition, select this checkbox.</td></tr>
<tr><td>Cert Sync*</td><td>Select from one of the following options:
• Managed: AppViewX will connect with the customer’s AWS account and discover certificates. These certificates will be added to the inventory. Users with the relevant permissions can then perform the required certificate-related actions.
• Monitored: AppViewX will connect with the customer’s AWS account and discover certificates. These certificates</td></tr>
</table>

Fields	Description
	will be added to the inventory where the users will be allowed to only view the certificates. • **Ignored**: AppViewX will connect with the customer's AWS account but certificate discovery will be disabled.
Auto Sync	To enable/disable automatic schedule-based synchronization: a. For **Auto Sync**, select the **Yes** checkbox. b. For **Schedule based discovery**, use the two dropdown lists to select a duration. For example, to schedule the auto sync after every 2 days, from the first dropdown list, select **2** and from the second dropdown list, select **Days**. By default, the auto sync is set to **1 Hours**. **Note:** The **Schedule based discovery** dropdown lists are displayed only when **Auto Sync** is enabled.
Route53 Zone Auto Approval	To support DNS validation as an automatic process, enable this toggle. **Important:** If Route53 has been configured for any of the older Amazon Public CAs, ensure that, after migration, the zones are manually updated.
*: *Mandatory fields*	

8. Click **Save**.

Configuring Amazon Private CA

1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Certificate Authority**.
2. From the displayed CA, Select **Amazon**.
   The **Amazon** home page is displayed.
3. To configure the Amazon Private CA, click **AWS Private CA**.

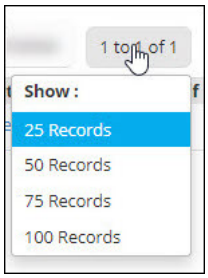


The **Amazon** home page is updated to display the inventory grid as shown in the image. In the inventory grid for the Amazon Private CA, master and child account details are logged as separate entries, instead of having just one master entry.

Fields in the inventory grid are explained in the table below:

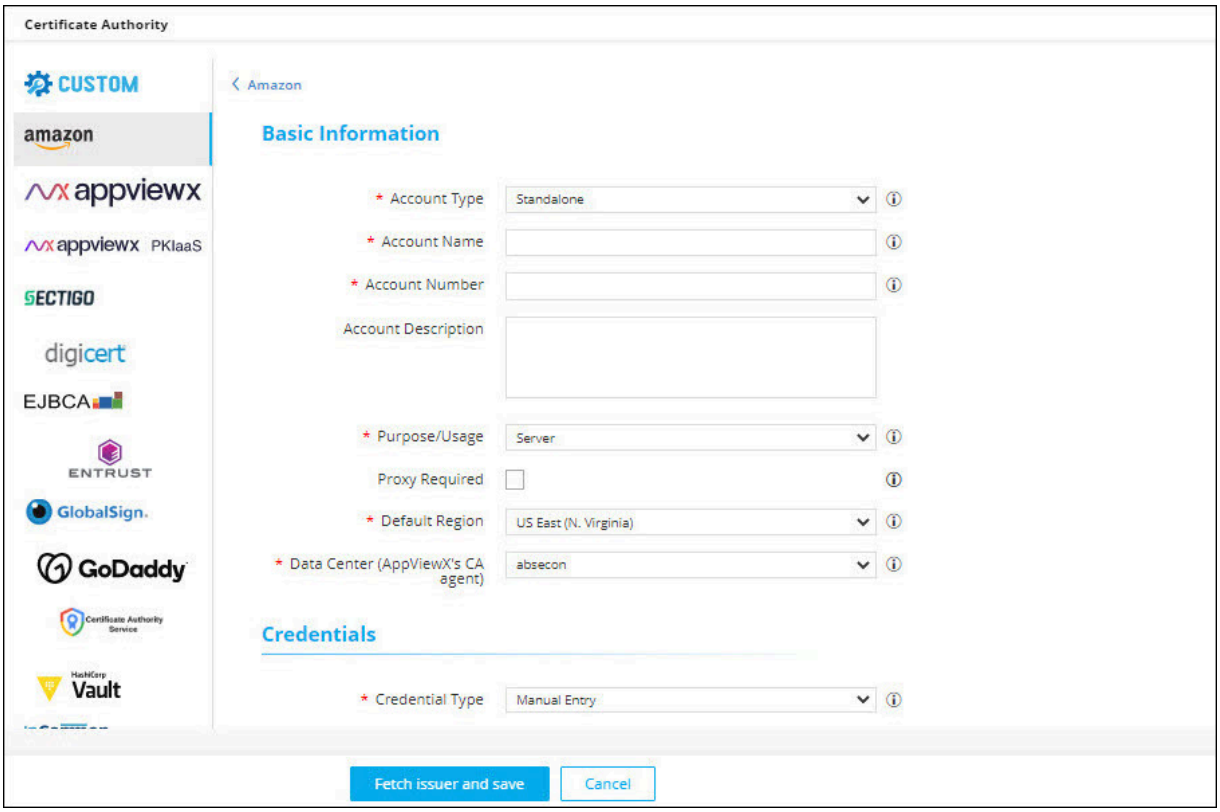
AWS Private CA - Screen Description Table

Fields	Description
Search	Use the **Search** field to search for accounts, by entering the value of one of the details listed in the inventory grid.
Delete	To delete one or more accounts: a. From the inventory grid, select the checkbox corresponding to the account(s) you want to delete. b. Click Delete. **Tip:** To delete all accounts listed in the inventory grid, select the checkbox in the grid header.

Fields	Description
	To set the number of records that should be displayed on one page: a. Click  . b. From the **Show** menu displayed, select the required value. 
	If the inventory grid spans more than one page, use this control to navigate the pages, one page at a time.
Account Name	This is the unique name for the Certificate Authority (CA) account entered at the time of account creation.
Account Number	AWS account number
Account Type	**Multi account**: Indicates that the account is a cross account **Single account**: Indicates that the account is a standalone account
CA Status	For an account, after all configuration details for Amazon Private CA are entered, you will be required to click the **Fetch issuer and save** button to sync and discover the issuers and the respective certificates for that account. The **CA Status** field shows the current status of this sync and discovery process. Possible values for this field are: • Completed • In progress

Fields	Description
	Note: An account entry in the grid will be disabled till the **CA Status** is **In progress**.
Connection Status	To validate if connection has been established with the CA, click **Check**. If a connection has been established, this field is updated to display **Success** or **Failure**.
No. of Issuers	This field displays the number of issuers associated with the account. **Note:** For a master account, this field will show the number of issuers associated with only the master account. The value does not include the number of issuers associated with the child account.
*: *Mandatory fields*	

4. Click the **Configure Now** button or **+Add** icon from the middle or top-right of the page respectively. The **Amazon** page is updated to display fields for entering the CA configuration-related information.



5. On this screen, enter the following **Basic Information**:

Basic Information - Field Description Table

Fields	Description
Account type*	From the dropdown list, from the following options, select the customer's AWS account type: • **Standalone**: The user account and the resources are available in the same account. • **Cross or Federated**: Resources are available across multiple accounts and users are given role-based access.
Account name*	Enter a unique name for the Certificate Authority (CA) account that will be used during certificate enrollment and policy creation.
Account number*	Enter the customer's AWS account number.
Account Description	Enter any additional details related to the account, if required.
Purpose/Usage*	From the dropdown list, select the purpose of the certificate that can be requested using this account.

Fields	Description
Proxy Required	To allow all communication to the Certificate Authority (CA) to use the proxy details (provided in general settings; refer the CLMaaS Platform User Guide for more details), select this checkbox.
Default Region*	From the dropdown list, select the default region for API communication.
Data Center (AppViewX's CA Agent)	From the dropdown list, select the data center that will be used to establish communication with the Certificate Authority (CA)
*: *Mandatory fields*	

6. Enter the following **Credentials**-related information:

Credentials - Field Description Table

Fields	Description
Credential type*	From the dropdown list, from the following options, select the credential type: • **Manual Entry**: Manually enter the access and secret key for the customer's AWS account)
Access key*	Enter the access key ID for the customer's AWS account. The access key and the secret access key (entered in the following field) are used together to authenticate requests. **Note:** This field is displayed only when **Credential** type is set to **Manual Entry**.
Secret key*	Enter the secret access key ID for the customer's AWS account. The access key (entered in the previous field) and the secret access key are used together to authenticate requests.

Fields	Description
	Note: This field is displayed only when **Credential** type is set to **Manual Entry**.
Credential name*	If the customer's AWS credentials are stored in CyberArk, from the dropdown list, select the CyberArk credential name. **Note:** This field is displayed only when **Credential** type is set to **Credential List - CyberArk**.
*: *Mandatory fields*	

7. In the **Discover resources** section, enter the following details:

Discover Resources - Field Description Table

Fields	Description
Role ARN for Resource Discovery*	**Note:** This field is displayed only when **Account Type** is **Cross or Federated**. To let the master account assume role for the child account (get temporary privileges to discover resources from the child account), configure the role ARN for resource discovery: a. Click  . b. Enter the following details:

Fields	Description
Role Session name	**Role Session name** is an identifier for the assumed role session. Use the **Role Session name** to uniquely identify a session when the same

Fields	Description	
	Fields	Description
		rule is assumed by different principals or for different reasons.
	Duration Seconds	Enter the duration, in seconds, for which the credentials should remain valid. Acceptable durations for IAM user sessions: • **Minimum**: 900 seconds (15 minutes) • **Maximum**: 129,600 seconds (36 hours) **Default**: 3600 seconds (1 hour)
	External Id	**External Id** is a unique identifier that might be required when you assume a role in another account.
	Source Identity	The source identity is specified by the principal that is calling the **AssumeRole** operation.
	Session Tags	**Session Tags** are key-value pairs that you pass when you assume an IAM role or federate a user in AWS STS.

<table>
<tr><th>Fields</th><th>Description</th></tr>
<tr><td></td><td>
<table>
<tr><th>Fields</th><th>Description</th></tr>
<tr><td></td><td>To create a session tag:
i. In the Enter Key field, enter a key for the key-value pair.
ii. In the Enter Value field, enter a value for the key-value pair.
iii. Click Add.
The added key-value pair is shown in the table below the fields.</td></tr>
</table>
</td></tr>
<tr><td>Service Region*</td><td>Service regions are regions that are supported by the selected service.

To select a service region:
a. To fetch the service regions for the account information provided, click Fetch Region.The retrieved service regions are populated in the Select the Region(s) dropdown list.
b. From the Select the Region(s) dropdown list, select the required service region.</td></tr>
<tr><td>CA Operation Mode*</td><td>From the following options, select one/both operation mode(s) for discovering all the certificates enrolled by the Private Certificate Authority:

• ACM Private CA

• AWS Certificate Manager (ACM)</td></tr>
</table>

<table>
<tr><th>Fields</th><th>Description</th></tr>
<tr><td>S3 Bucket*</td><td>
Note: This field is displayed only when the ACM Private CA operation mode is selected.

Enter the S3 bucket name.
</td></tr>
<tr><td>Role ARN for S3 Bucket</td><td>
Note: This field is displayed only when the ACM Private CA operation mode is selected for a Cross or Federated account.

a. Click  .

The ARN Advanced Settings action pane is displayed.

b. In the ARN Advanced Settings action pane, enter the following details:
<table>
<tr><th>Fields</th><th>Description</th></tr>
<tr><td>Role Session name*</td><td>Role Session name is an identifier for the assumed role session.

Use the Role Session name to uniquely identify a session when the same rule is assumed by different principals or for different reasons.</td></tr>
<tr><td>Duration Seconds</td><td>Enter the duration, in seconds, for which the credentials should remain valid.</td></tr>
</table>
</td></tr>
</table>

Fields	Description
	<table><tr><th>Fields</th><th>Description</th></tr><tr><td></td><td>Acceptable durations for IAM user sessions: • **Minimum**: 900 seconds (15 minutes) • **Maximum**: 129,600 seconds (36 hours) **Default**: 3600 seconds (1 hour)</td></tr><tr><td>**External Id**</td><td>**External Id** is a unique identifier that might be required when you assume a role in another account.</td></tr><tr><td>**Source Identity**</td><td>The source identity is specified by the principal that is calling the **AssumeRole** operation.</td></tr><tr><td>**Session Tags**</td><td>**Session Tags** are key-value pairs that you pass when you assume an IAM role or federate a user in AWS STS.</td></tr></table>

<table>
<tr><th>Fields</th><th>Description</th></tr>
<tr><td></td><td>
<table>
<tr><th>Fields</th><th>Description</th></tr>
<tr><td></td><td>To create a session tag:
i. In the Enter Key field, enter a key for the key-value pair.
ii. In the Enter Value field, enter a value for the key-value pair.
iii. Click Add.
The added key-value pair is shown in the table below the fields.</td></tr>
</table>
c. Click Apply.
</td></tr>
<tr><td>Discover Certificate</td><td>To enable instant certificate discovery at the time of device addition, select this checkbox.</td></tr>
<tr><td>CA Sync*</td><td>Select from one of the following options:
• Managed: AppViewX will connect with the customer’s AWS account and discover certificates. These certificates will be added to the inventory. Users with the relevant permissions can then perform the required certificate-related actions.
• Monitored: AppViewX will connect with the customer’s AWS account and discover certificates. These certificates will be added to the inventory where the users will be allowed to only view the certificates.
• Ignored: AppViewX will connect with the customer’s AWS account but certificate discovery will be disabled.</td></tr>
</table>

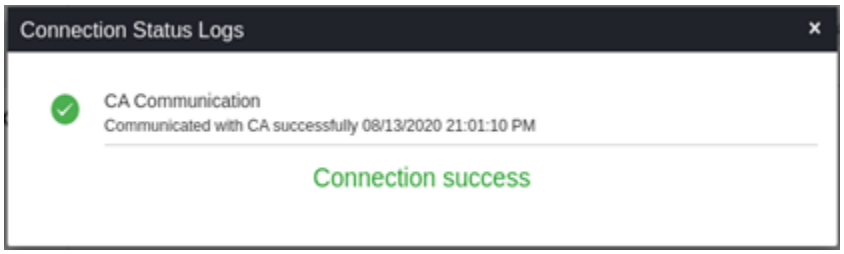
Fields	Description
Auto Sync	To enable/disable automatic synchronization, use the **Auto Sync** key. If **Auto Sync** is enabled, to set the frequency of the schedule-based sync: a. From the first dropdown list, select the interval between two schedule-based syncs. b. From the second dropdown, select a unit for the interval (**Hours**/**Days**). For example, to set the frequency of the schedule-based sync to every 2 hours, from the first dropdown list, select **2** and from the second dropdown list, select **Hours**.
*: *Mandatory fields*	

8. Click **Fetch issuer and save**.
   - AppViewX will now discover all the Private CA Certificate Authorities across the selected region(s).
   - The inventory grid on the Amazon CA home page will be populated with the properties and details retrieved from this discovery.

Validating Amazon

Once the Amazon settings are added, you need to validate the connection between AppViewX and Amazon, to make sure that the connection is properly configured.

1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Certificate Authority**.
2. From the displayed CA, Select **Amazon**.
   The **Amazon** home page is displayed.
3. On the **Amazon** home page, select **Amazon** or **Amazon Private CA**.
4. In the Status column of the grid with the listed accounts, click **Check** to validate the CA setting that is created.



The CA communication will be validated and the **connection status** will be displayed as either **Connection success** or **Failure**.

CSC Global CA

Prerequisites

The following values are required to configure the CSC Global CA:

- CA Base URL (the value is available in the UI field of the CA setting page)
- API Key
- Token
- AppViewX server should either have internet access or have a proxy configured in AppViewX general settings. Refer to the section Managing Proxy Settings in the Platform guides.

For API Key and Token, follow the steps below:

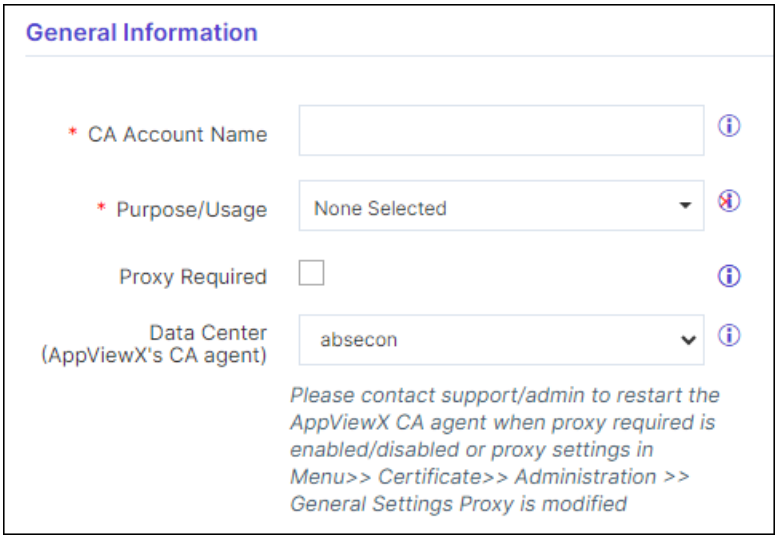
1. Login to https://weblogin.cscglobal.com/public/login and click **CSC Domain Manager** (you will see the CSC Domain Manager page after the 2-factor authentication).
2. On the top navigation bar, click **API** > **API Administration**.
3. The organization account has the common API **Key** displayed above the user-list table. Click **Show** to view and copy the API key.

   Multiple users can use the same API key, but each user has an individual token that admins/users can create.
4. If a first time user or users with an expired token, click **Create Token**.
   The token is displayed in a pop-up only once at the time of creation. Save and copy for further use.

> **Note:** Current users can rotate/renew/ or create new tokens (this can only be done until the current token is valid) using the *Refresh Expired Token* API.

Configuring CSC Global CA

1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Certificate Authority**.
2. From the displayed CA, Select **CSC**.
   The **CSC** home page is displayed.
3. Click the **Configure Now** button or **+Add** icon from the middle or top-right of the page respectively.
   The **CSC Global CA** configuration page is displayed.
4. Update the following details in the **General Information** section as described in the table:

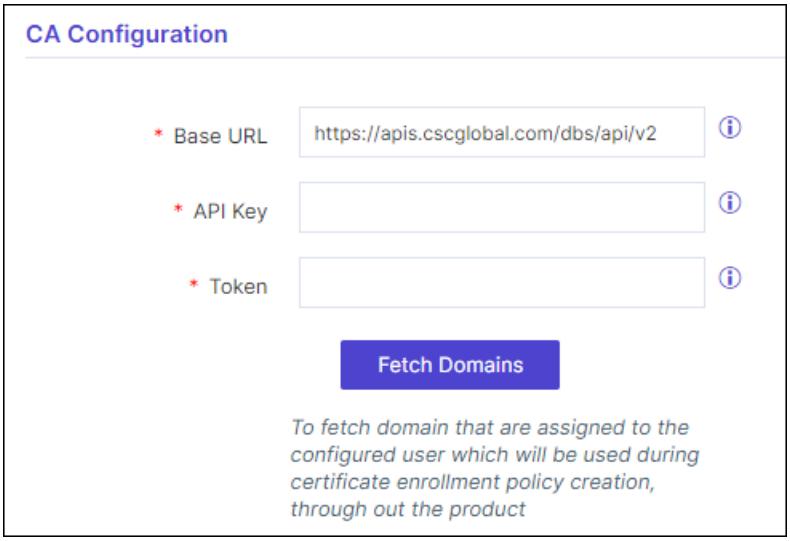


General Information - Field Description Table

Fields	Description
***CA Account name**	Enter a unique name in the text field to identify the CA account and be represented during certificate enrollment and policy creation. No special characters other than '.', '-','_' are allowed. Names should not start with special characters.
***Purpose/Usage**	Select the purpose of the certificate from the dropdown list for which CLM actions will be enabled. Example: Server, Client
Proxy Required	Enable this field if the CA communication needs to happen via Proxy. The proxy details configured in general settings will be used for communication.
Data Center (AppViewX's CA agent)	Select the data center through which the CA communication needs to happen.

Fields	Description
*: *Mandatory fields*	

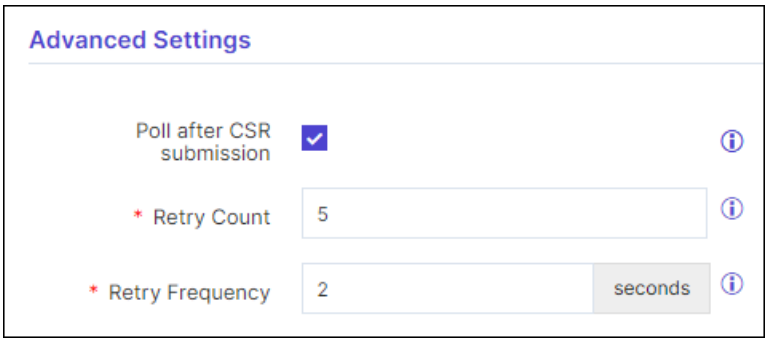
5. Update the following details in the **CA Configuration** section as described in the table. These fields are necessary for invoking the CSC Global CA APIs for Certificate Management:



CA Configuration - Field Description Table

Fields	Description
***CA Base URL**	Enter the base URL that will be used to construct the API request. The value is *https://apis.cscglobal.com/dbs/api/v2*
***API Key**	Enter the API key which is a unique identifier used to authenticate the user. The value entered will be masked and encrypted.
Token	Enter the API token which is used to authenticate users or applications and provide access to protected resources. The value entered will be masked and encrypted.
*: *Mandatory fields*	

6. Click **Fetch Domain.**
   If the data provided is accurate the success message “Domains fetched successfully“ will be displayed and a table is displayed with two columns - *Account Number* and *Domain Name*. A list of active domains in AppViewX are listed in the table. Each account can hold multiple domains.
7. Update the following details in the **Advanced Settings** section as described in the table.



Advanced Settings - Field Description Table

Fields	Description
***Poll after CSR submission**	Selecting the checkbox enables polling after CSR submission. It fetches the certificates immediately after CSR submission on enrollment, renew, and reissue with the retry count and retry frequency specified in the fields below.
***Retry Count**	Enter a value between 1 - 10.
***Retry Frequency**	Enter a value between 1 - 30 seconds.
*: *Mandatory fields*	

8. Click **Save**.
   The CA details are saved and the **Fetch Custom Fields button** is enabled.

> **Note:** In case “Fetch Domains” and “Fetch Custom Fields” are not triggered before saving CA settings, this action can explicitly perform both these actions and update the CA.

9. To get the list of custom fields configured (if any), click **Fetch Custom Fields**.
   If the custom fields are configured they will be displayed in a table with two columns - *Name* and *Mandatory/Optional*.
   a. Enter the values for the custom fields.
   b. Click **Save**.

   The details are saved.

> **Note:** These custom fields attributes are displayed in the certificate enrollment form at the time of enrollment.

Validating CSC Global

Once the CSC Global CA settings are added, validation needs to be done to check whether the connection between AppViewX and CSC is properly configured.

1. On the CSC CA page, the above configured CA will be displayed with a **Check** button in the last column of the table.
2. Click **Check** to validate the CA setting that is created.
   The CA communication will be validated and the **Connection Status** will be shown as either **Success** or **Failure**.

Custom CA

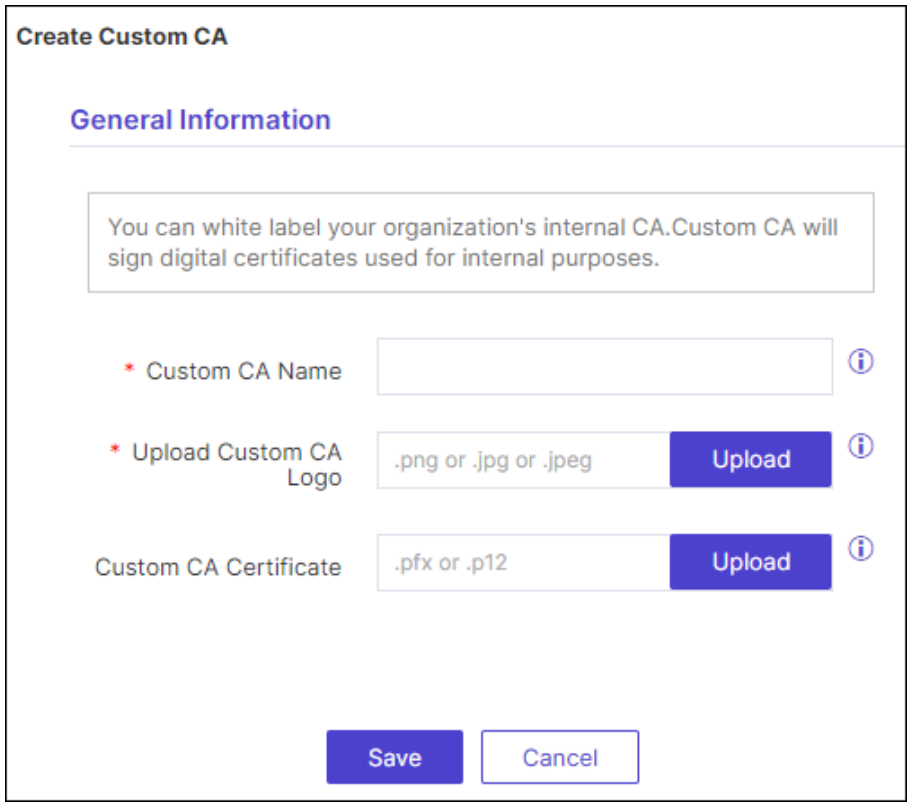
Prerequisites

The prerequisites for configuring Custom CA account in AppViewX are as follows:

- A logo to use it for the custom CA.
- An optional CA certificate and key to be used as a root certificate.

Configuring Custom CA

1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Certificate Authority**.
2. From the displayed CA, Select **Custom**.
3. Click the **Configure Now** button or **+Add** icon from the middle or top-right of the page respectively.
   The **Custom** home page is displayed.



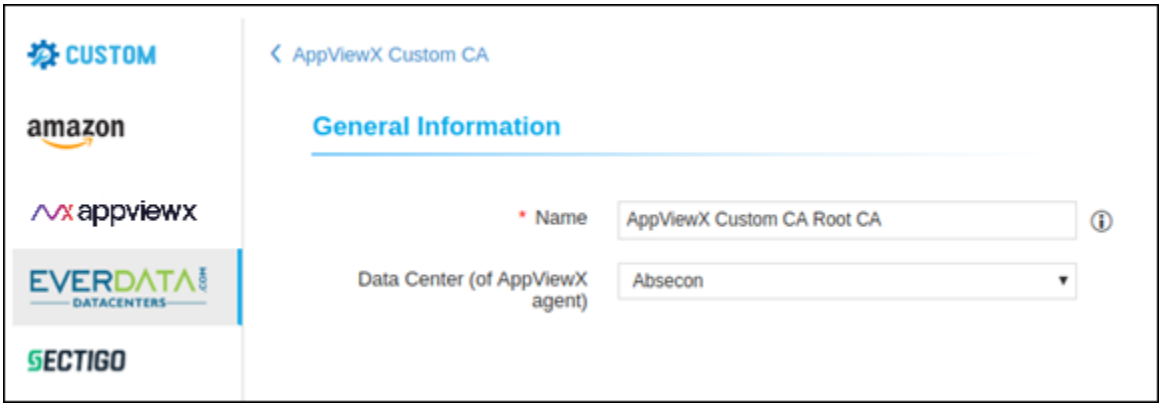
4. Update the following details in the **General Information** section as described in the table:

General Information - Field Description Table

Fields	Description
***Custom CA Name**	A unique name to identify the CA name. **Note:** No special characters allowed.
***Upload Custom CA Logo**	Upload a logo for the custom CA. This logo will appear in the product representing the custom CA.
Custom CA Certificate	Upload a certificate for the custom CA. This certificate will become the root certificate. **Note:** The <.*pfx*> and <.*p12*> are certificate types are supported.
*: *Mandatory fields*	

5. Once the logo and certificate are uploaded, the entered CA will appear in the CA list with the logo presented.

6. Once the logo is added, users can click **Configure Now** to input the CA details.
7. Update the following details in the **General Information** section as described in the table:



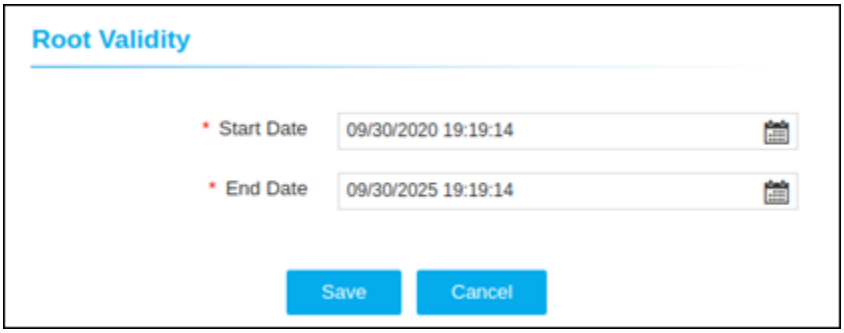
Fields	Description
***Name**	Client authentication certificate for API communication.
Data Center (AppViewX's CA agent)	Select the data center through which the CA communication needs to happen.
*: *Mandatory fields*	

8. Update the following details in the **ROOT CSR parameters** section as described in the table:

Root CSR - Field Description Table

Fields	Description
Common Name	The common name of the root certificate. **Note:** • Use Asterisk (*) for the host part of the FQDN to enforce the domain. For example, *.domain.com will only allow users to request certificates with domain domain.com. • Allowed Special Characters: Asterisk (*), Hyphen (-), Period (.)
Algorithm	Type of the root certificate.
Hash Function	The hash function for the root certificate.
Organization Unit	Name of the Organisation unit.
Key Length	Key length for the root certificate.
Organization	Organization attribute for the root certificate.
Locality	Locality attribute for the root certificate.
State or Province	State attribute for the root certificate.
Country	Country attribute for the root certificate.
Email Address	Email address for the root certificate.
*: *Mandatory fields*	

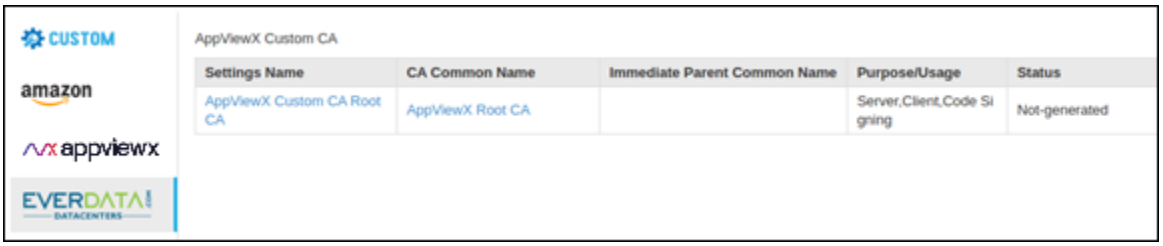
9. Update the following details in the **Root Validity** section as described in the table.



Fields	Description
***Start Date**	Start date of the certificate issuance.
***End Date**	End date of the certificate issuance.
*: *Mandatory fields*	

10. Click **Save**.
    Once the setting is saved, the user will be directed to the root certificate submission holistic view as below.

11. Users can submit and fetch the root certificate.
12. On the CA setting page user can see the status of the created setting as shown below.



DigiCert CA

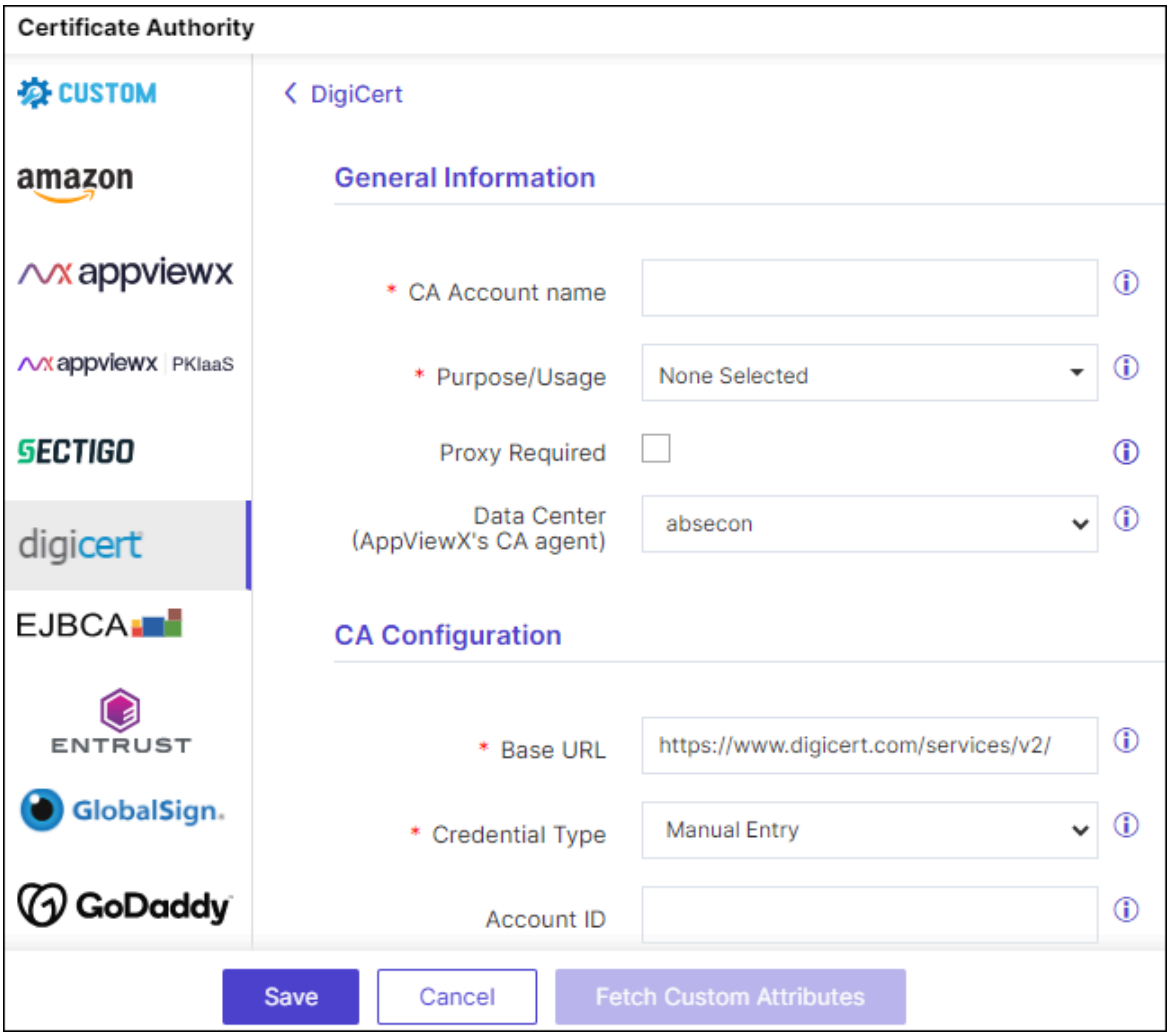
Prerequisites

The prerequisites for configuring DigiCert CA account in AppViewX are as follows:

- A DigiCert CertCentral Account with **Administrator** role Access.
- An **API Key** configured in Digicert with required permissions to make API Requests from AppViewX.
- AppViewX server should either have internet access or have a proxy configured in AppViewX general settings.

Configuring Digicert

1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Certificate Authority**.
2. From the displayed CA, Select **DigiCert**.
   The **DigiCert** home page is displayed.
3. Click the **Configure Now** button or **+Add** icon from the middle or top-right of the page respectively.
   The **DigiCert** configuration page is displayed.



4. Update the following details in the **General Information** section as described in the table:

General Information - Field Description Table

Fields	Description
***CA Account name**	A unique name to identify the CA setting. **Note**: No special characters other than '.', '-','_' are allowed. Names should not start with special characters.
***Purpose/Usage**	Certificate Type for which CLM actions will be enabled. Example: Server, Client
Proxy Required	Enable this field if the CA communication needs to happen via Proxy. The proxy details configured in general settings will be used for communication.
Data Center (AppViewX's CA agent)	Select the data center through which the CA communication needs to happen.
*: *Mandatory fields*	

5. Update the following details in the **CA Configuration** section as described in the table. These fields are necessary for invoking the Digicert CA APIs for Certificate Management:

CA Configuration - Field Description Table

Fields	Description
***Base URL**	This URL will contain just the hostname of the Digicert CA instance. For example, <https://www.digicert.com> **Note:** vendorSpecificSettings.url - invalid URL.
***Credential Type**	Select the type of credential as desired from the dropdown list. The available options are, • Manual EntryCredential • List - CyberArk.
***Credential List**	Select the required credential from the dropdown list.

Fields	Description
	Note: This field will be enabled if the Credential Type is selected as Credential List - CyberArk.
Account ID	Account id details of Digicert CA Account, which can be found under account manager details in Digicert CertCentral Account.
***API Key**	API key specific to the CA account. This API key should have required permission to make API Calls. Space is not allowed.
Auto Approve	Enable **the Auto Approve** option if all CLM requests from AppViewX do not need to be approved from Digicert CA Account.
*: *Mandatory fields*	
Note: Auto approval checkbox is optional and its features work only for **one-step certificate requests** configured in the Digicert Cert Central Account.	

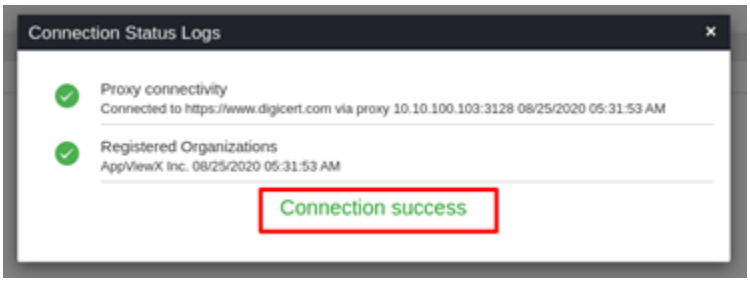
6. Select **Fetch Divisions and Certificate Types.**
   The Division and Certificate types available in the DigiCert CA account will be fetched.
7. Click **Save**.

Validating Digicert

Once the Digicert settings are added, the validation must be done to check whether the connection between AppViewX and Digicert is configured properly.

1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Certificate Authority**.
2. From the displayed CA, Select **Digicert**.
   The **Digicert** home page is displayed.
3. In the Status column of the grid with the listed accounts, click **Check** to validate the CA setting that is created.

The CA communication will be validated and the **Connection Status** will be shown as either **Success** or **Failure**.



DigiCert MPKI

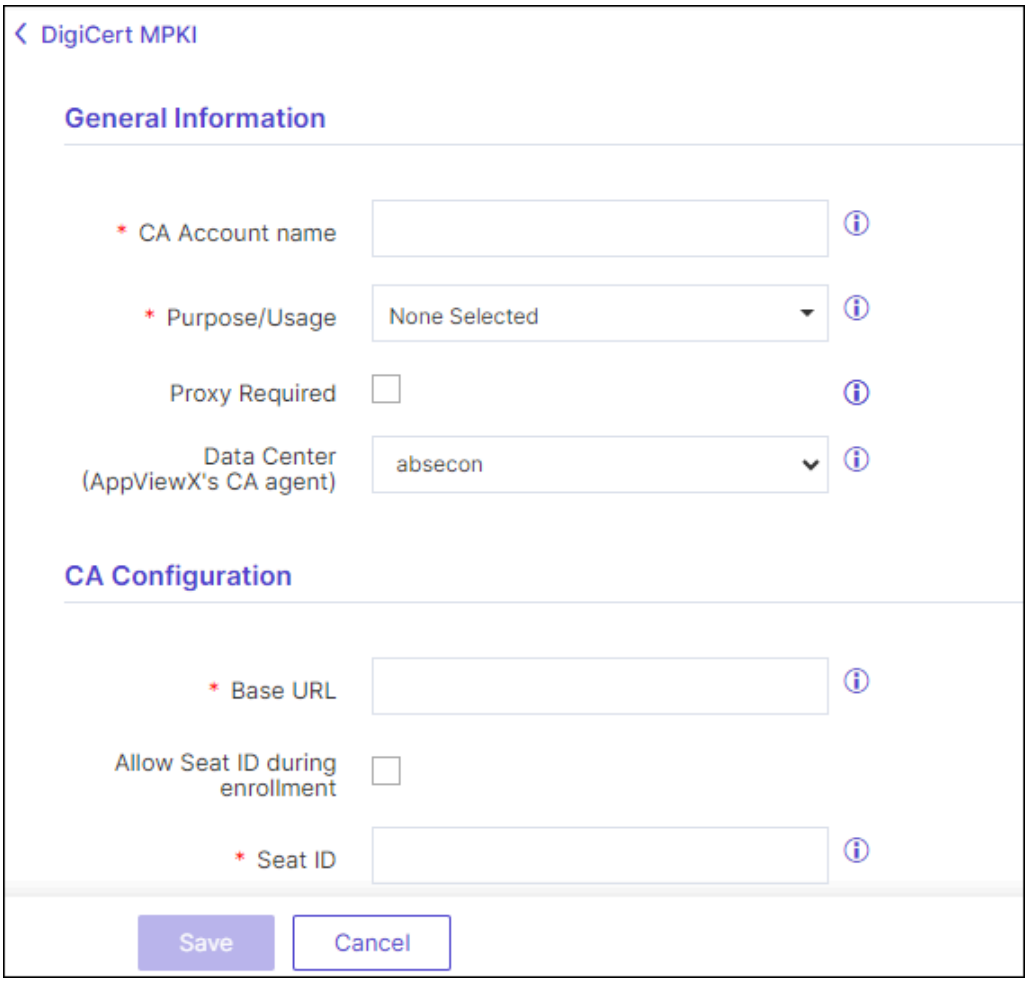
Prerequisites

The prerequisites for configuring a Digicert MPKI CA account in AppViewX are as follows:

- A Digicert MPKI Account with **Administrator** role Access.
- An **API Key** configured in Digicert MPKI with required permissions to make API Requests from AppViewX.
- The AppViewX server should either have internet access or have a proxy configured in AppViewX general settings.

Configuring DigiCert MPKI

1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Certificate Authority**.
2. From the displayed CA, Select **DigiCert**.
   The **DigiCert** home page is displayed.
3. Click the **DigiCert MPKI** tab.
4. Click the **Configure Now** button or **+Add** icon from the middle or top-right of the page respectively.
   The **DigiCert MPKI** configuration page is displayed.



5. Update the following details in the **General Information** section as described in the table.

General Information - Field Description Table

Fields	Description
***CA Account name**	A unique name to identify the CA setting. No special characters other than '.', '-','_' are allowed. Names should not start with special characters.
***Purpose/Usage**	Certificate Type for which CLM actions will be enabled. Example: Server, Client
Proxy Required	Enable this field if the CA communication needs to happen via Proxy. The proxy details configured in general settings will be used for communication.

Fields	Description
Data Center (AppViewX's CA agent)	Select the data center through which the CA communication needs to happen.
*: *Mandatory fields*	

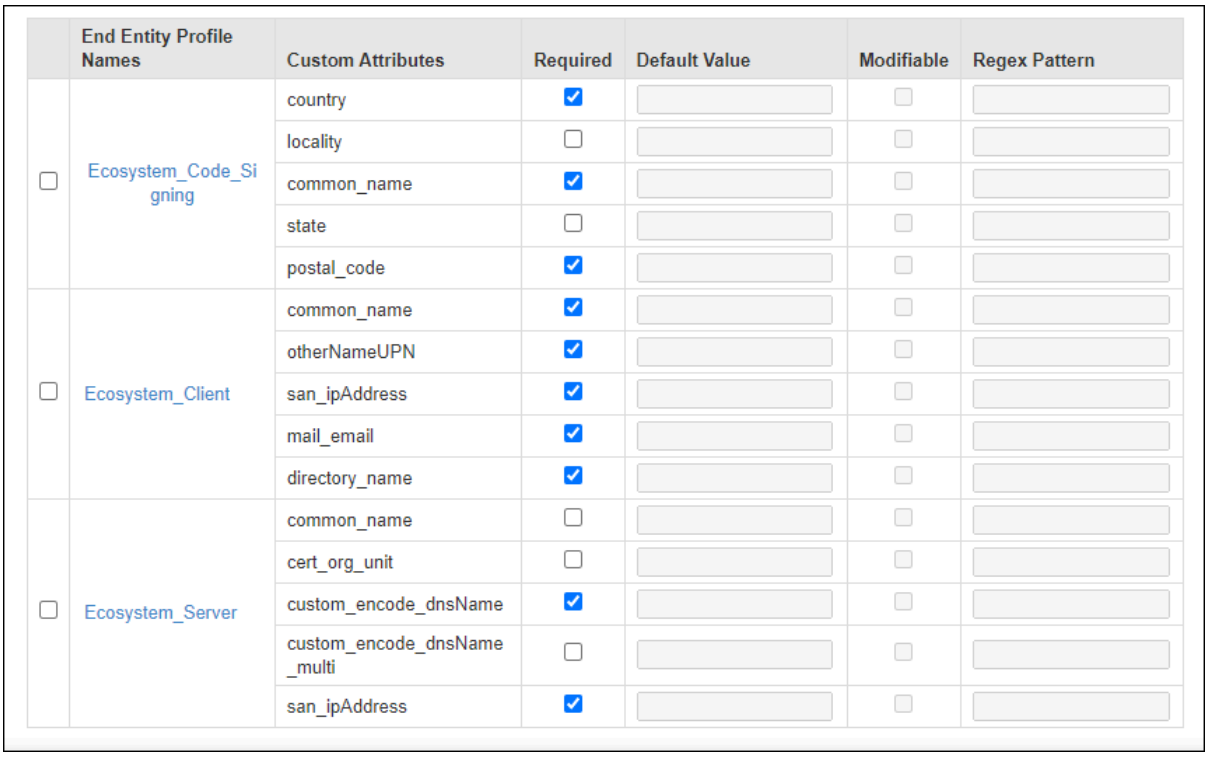
6. Update the following details in the **CA Configuration** section as described in the table. These fields are necessary for invoking the Digicert CA APIs for Certificate Management.

CA Configuration - Field Description Table

Fields	Description
***Base URL**	This URL will contain just the hostname of the Digicert CA instance. For example, <https://www.digicert.com>
***Seat ID**	Enter the unique seat id for discovering certificates.
***API Key**	Enter the API key, which is generic across all CAs
*: *Mandatory fields*	

Note: Auto approval checkbox is optional and features work only for **one-step certificate requests** configured in the DigiCert Central Account.

7. Select **Fetch Divisions and Certificate Types.**
   The Division and Certificate types available in the Digicert CA account will be fetched and listed for the specific API key user in the table as shown below.



	End Entity Profile Names	Custom Attributes	Required	Default Value	Modifiable	Regex Pattern
☐	Ecosystem_Code_Signing	country	☑		☐	
		locality	☐		☐	
		common_name	☑		☐	
		state	☐		☐	
		postal_code	☑		☐	
☐	Ecosystem_Client	common_name	☑		☐	
		otherNameUPN	☑		☐	
		san_ipAddress	☑		☐	
		mail_email	☑		☐	
		directory_name	☑		☐	
☐	Ecosystem_Server	common_name	☐		☐	
		cert_org_unit	☐		☐	
		custom_encode_dnsName	☑		☐	
		custom_encode_dnsName_multi	☐		☐	
		san_ipAddress	☑		☐	

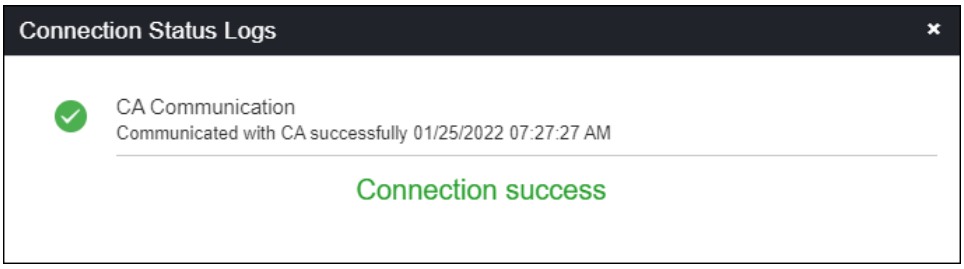
8. Click **Save**.
   A pop-up message is displayed as <CA_name> Settings Added.

Validating DigiCert MPKI Connection

Once the Digicert MPKI settings are added, the validation must be done to check whether the connection between AppViewX and Digicert MPKI is configured properly.

1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Certificate Authority**.
2. From the displayed CA, Select **DigiCert**.
   The **DigiCert** home page is displayed.
3. Click **DigiCert MPKI** from the left pane of the page.
   The **DigiCert MPKI** home page is displayed.
4. In the Status column of the grid with the listed accounts, click **Check** to validate the CA setting that is created.

The CA communication will be validated and the **Connection Status** will be shown as either **Success** or **Failure**.



DigiCert One

About DigiCert One

DigiCert ONE is a modern PKI platform that provides a scalable foundation for fast and flexible PKI deployments. It provides an interface for managing your certificates and devices, customizing and automating workflows, and integrating DigiCert ONE with your existing PKI management tools.

In AppViewX's implementation of DigiCert One, we integrate with a key DigiCert One component, called the **Trust Lifecycle Manager**, which is used to perform certificate lifecycle management, discovery, notification, and automation.

Prerequisites

In order to configure DigiCert One CA account you will need the following:

- DigiCert One account base URL
- API Key or Client Authentication certificate (depending on the authentication mode)

Configuring a DigiCert One CA Account

1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Certificate Authority**.
   The **Certificate Authority** page is displayed.
2. On the **Certificate Authority** page, from the CA list displayed in the left, select **DigiCert**.
   The **Certificate Authority** page for **DigiCert** is displayed.
3. Click the **DigiCert One** tab.
4. To create your first DigiCert One CA account, click **Configure Now**.
   OR

   Click **+Add**.

The **Certificate Authority** page is updated to display the form fields for configuring a DigiCert One CA account.

5. Enter/Select the **General Information** for the CA account.

General Information - Field Description Table

Fields	Description
***CA Account name**	A unique name to identify the CA setting. No special characters other than '.', '-','_' are allowed. Names should not start with special characters.
***Purpose/Usage**	Certificate Type for which CLM actions will be enabled Example: Server, Client
Proxy Required	Enable this field if the CA communication needs to happen via a proxy. The proxy details configured in general settings will be used for communication.
Data Center (AppViewX's CA agent)	Select the data center through which the CA communication needs to happen.
*: *Mandatory fields*	

6. Enter/Select the **CA Configuration** details.

General Information - Field Description Table

Fields	Description
***Base URL**	Hostname of DigiCert One instance. For example http://one.digicert.com
***Authentication method**	From the following options, select an authentication method for authenticating the API requests: • **API Token**: To authenticate the API requests with an API token, select this option. • **Client Certificate**: To authenticate the API requests using a client certificate file, select this option. **Important:** Currently, only the **API Token** authentication method is supported.

Fields	Description
***API Key**	Enter the API key that will be used for authentication. **Note:** You will be assigned an API key when you login to your DigiCert One instance and create a user.
Allow Seat ID during enrollment	Seat ID is a unique user-defined value assigned to identify an entity in the DigiCert One account. The seat ID for a certificate is used for certificate enrollment, renewal, and regeneration. For the certificates enrolled for this CA account, you can either assign a unique seat ID for each certificate or a common one for all the certificates. To assign a unique Seat ID for the certificates enrolled for this CA account, select this checkbox. If you select this option, a field to specify the seat ID is hidden from the CA settings form and is included in the certificate enrollment form.
***Seat ID**	Seat ID is a unique user-defined value assigned to identify an entity in the DigiCert One account. The seat ID for a certificate is used for certificate enrollment, renewal, and regeneration. To have a common Seat ID for all certificates enrolled for this CA account.
Use DigiCert One to switch certificates from DigiCert MPKI	To automatically switch your DigiCert MPKI certificates to DigiCert One at the time of auto regeneration, select this checkbox. **Important:** For multiple CA accounts with automatic CA switch enabled, CA switch will use CA settings configured for the first CA account for which the automatic CA switch is enabled.

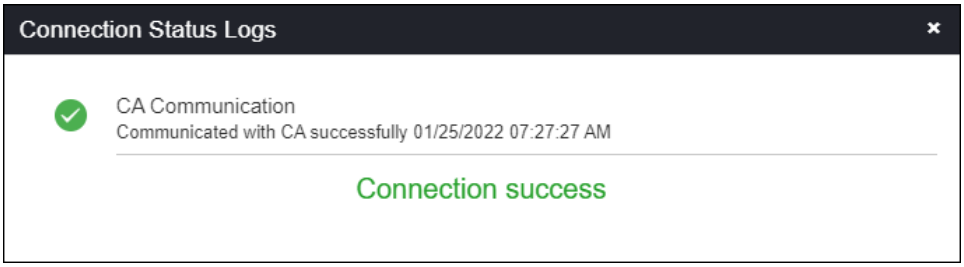
Fields	Description
	Note: To manually switch CAs, refer to the instructions here.
*: *Mandatory fields*	

7. To fetch profiles that are assigned to the configured user which will be used during certificate enrollment, policy creation, through out the product, click **Fetch Certificate Profiles**.
   All certificate profile configured with “enrollment_method”: **rest_api** and “authentication_method”: **third_party_app** are displayed.
8. Click **Save.**
   The CA account is listed in the inventory table.

Validating the DigiCert One CA Account Connection

Once the Digicert One CA settings are added, to validate that the connection between AppViewX and Digicert One is configured correctly:

1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Certificate Authority**.
   The **Certificate Authority** page is displayed.
2. On the **Certificate Authority** page, from the CA list displayed in the left, select **DigiCert**.
   The **Certificate Authority** page for **DigiCert** is displayed.
3. Click the **DigiCert One** tab.
   All existing DigiCert CA accounts are listed in the inventory on this page.
4. From the **Connection Status** column, click **Check**.
   The CA communication is validated and a success/failure message is displayed.



EJBCA CA

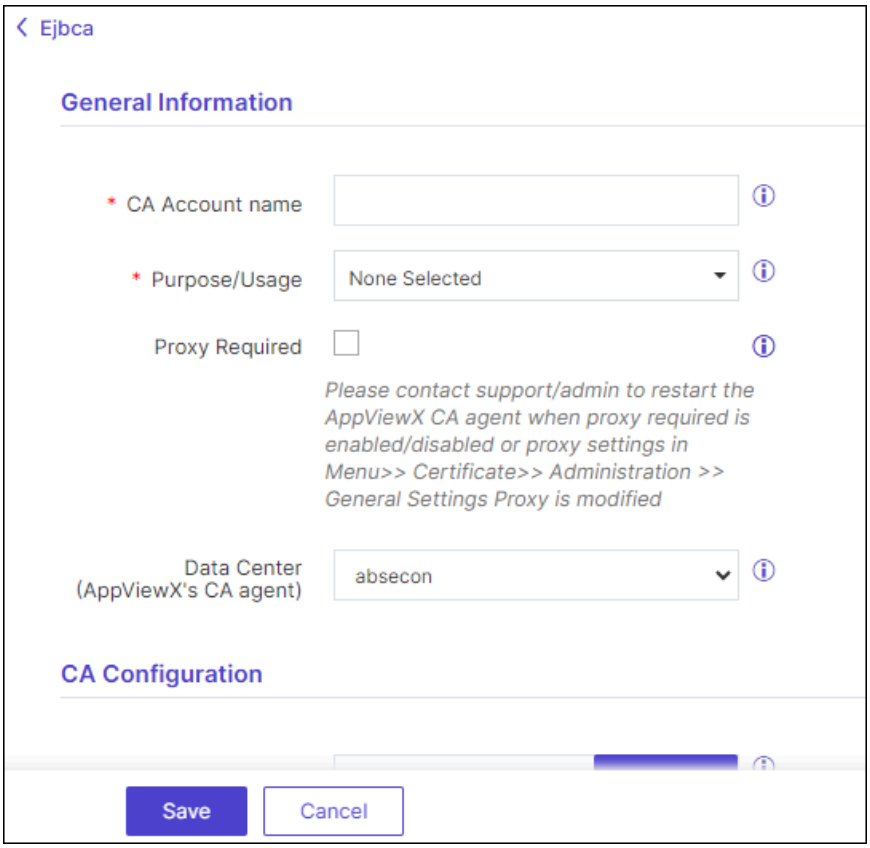
Prerequisites

The prerequisites for configuring the EJBCA account in AppViewX are as follows:

- An Ejbca client certificate for a user having the necessary access for enrolling the certificates and other CLM operations.
- AppViewX server should either have internet access or have a proxy configured in AppViewX general settings.

Configuring EJBCA

1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Certificate Authority**.
2. From the displayed CA, Select **EJBCA**.
   The **EJBCA** home page is displayed.
3. Click the **Configure Now** button or **+Add** icon from the middle or top-right of the page respectively.
   The **Ejbca** configuration page is displayed.



4. Update the following details in the **General Information** section as described in the table:

General Information - Field Description Table

Fields	Description
***CA Account name**	A unique name to identify the CA setting. **Note**: No special characters other than '.', '-','_' are allowed. Names should not start with special characters.
***Purpose/ Usage**	Certificate Type for which CLM actions will be enabled. E.g. Server, Client.
Proxy Required	Enable this field if the CA communication needs to happen via Proxy. The proxy details configured in general settings will be used for communication.
Data Center (AppViewX's CA agent)	Select the data center through which the CA communication needs to happen.
*: *Mandatory fields*	

5. Update the following details in the **CA Configuration** section as described in the table. These fields are necessary for invoking the APIs for Certificate Management.

CA Configuration - Field Description Table

Fields	Description
***Client Authentication**	Client authentication certificate for API communication. • Enter the valid password once the **Authentication Details** window is displayed. • Click **OK**. **Note:** Must be a valid <.*p12*> or <.*pfx*> file.
***URL**	Ejbca URL
***Discover by expiry days**	To get all the certificates that are expired and valid for specified days. **Note:** Must be a number.
End entity profile names	Required end entity profiles for CA setting.

Fields	Description
Custom attributes	Required custom attributes for the specific end entity profile. **Note:** Validation can be added by the user in the regex box.
*: *Mandatory fields*	

6. Click **Validate and Fetch**.
   The **End entity profiles** available for the CA account will be fetched along with the certificate profile from the **Certificate Authority**.
7. Update the following details in the **Certificate Attributes** section as described in the table:

Fields	Description
***End Entry Profile Names**	Select the profile that is used in the certificate enrollment from the dropdown list.
Custom Attributes	Select the list attributes configured in CA to enroll certificates.
*: *Mandatory fields.* **Note:** Custom attributes should be configured as exactly as it is available in the Ejbca portal.	

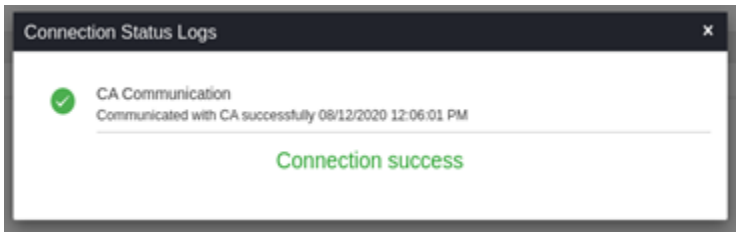
8. Click **Save**.

Validating EJBCA

Once the EJBCA settings are added, validation needs to be done to check whether the connection between AppViewX and EJBCA is properly configured.

1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Certificate Authority**.
2. From the displayed CA, Select **EJBCA**.
   The **EJBCA** home page is displayed.
3. In the Status column of the grid with the listed accounts, click **Check** to validate the CA setting that has been created.

The CA communication will be validated and the **Connection Status** will be shown as either **Success** or **Failure**.



Entrust CA

Prerequisites

The prerequisites for configuring Entrust CA account in AppViewX are as follows:

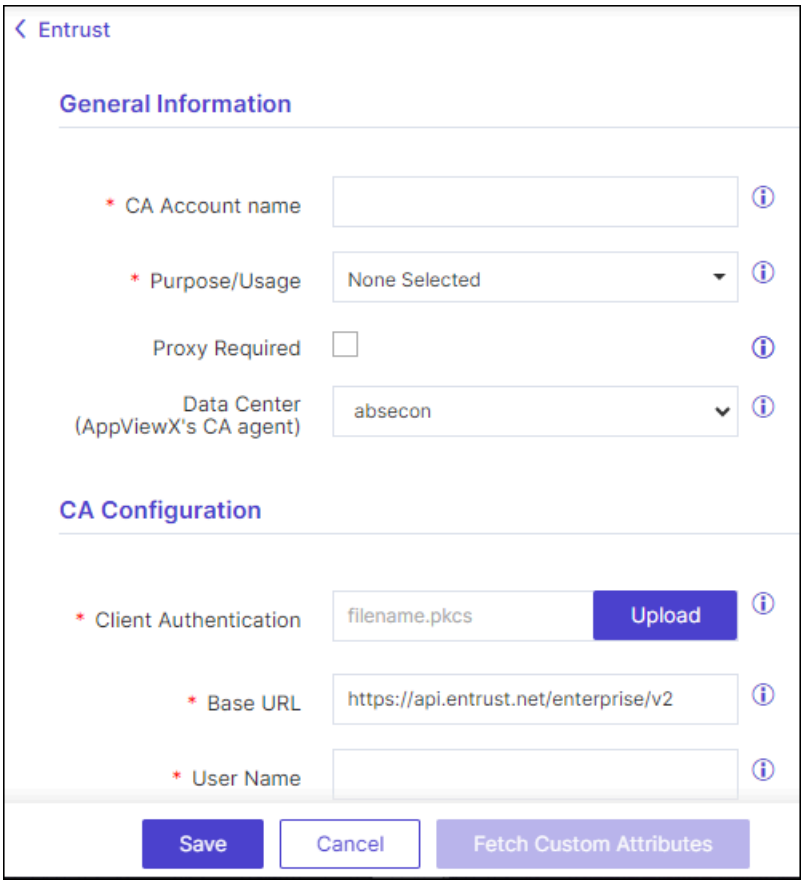
- An Entrust client authentication certificate and credentials−API username (username) and API key (password) having necessary access for CLM actions.

  To get a private key + certificate that can be used to access to API. The general steps performed by a super admin are:
  - Generate a new private TLS/SSL key and a CSR.
  - Issue a certificate in the ECS account using the CSR. Client Authentication must be enabled in the certificate.
  - Import the private key and certificate into the system that will be invoking the API.
  - Test that TLS/SSL mutual authentication is successfully configured.

  Refer chapter **Authentication** and section **TLS with client certificate authertication** in the Entrust - Rest API Guide.
- AppViewX server should either have internet access or have a proxy configured in AppViewX general settings.

Configuring Entrust

1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Certificate Authority**.
2. From the displayed CA, Select **Entrust**.
   The **Entrust** home page is displayed. The Entrust tab is selected by default.
3. Click the **Configure Now** button or **+Add** icon from the middle or top-right of the page respectively.
   The **Entrust** configuration page is displayed.



4. Update the following details in the **General Information** section as described in the table:

General Information - Field Description Table

Fields	Description
***CA Account name**	A unique name to identify the CA setting. **Note:** No special characters other than '.', '-','_' are allowed. Names should not start with special characters.
***Purpose/Usage**	Certificate Type for which CLM actions will be enabled. For example: Server and Client
Proxy Required	Enable this field if the CA communication needs to happen via Proxy. The proxy details configured in general settings will be used for communication.

Fields	Description
Data Center (AppViewX's CA agent)	Select the data center through which the CA communication needs to happen.
*: *Mandatory fields*	

5. Update the following details in the **CA Configuration** section as described in the table. These fields are necessary for invoking the Entrust CA APIs for Certificate Management.

CA Configuration - Field Description Table

Fields	Description
***Client Authentication**	The client authentication certificate from Entrust for API communication. **Note:** Must be a valid <.*p12*> file. To generate an CSR within AppViewX refer to Generating a CSR and download the CSR. Further, upload the CSR to the Entrust homepage as described in section - XXXXX.
***Base URL**	This URL will contain just the hostname of the Entrust CA instance. The value is https://api.entrust.net/enterprise/v2
User Name	Enter the API Username to communicate with the CA.
Password	Enter the API Password to communicate with the CA.
Auto Approve	Select the checkbox to avoid queuing of new certificates in the CA portal.
*: *Mandatory fields*	

6. Update the following details in the **Advanced Settings** section as described in the table.

Advanced Settings - Field Description Table

Fields	Description
Poll after CSR Submission	A check box field when selected will fetch the certificated immediately after CSR Submission on enrollment, renew, and reissue of certificate with the retry count and retry frequency as described below.
***Retry Count**	The number of times the polling will take place after CSR submission. Enter a value between 1 and 10.

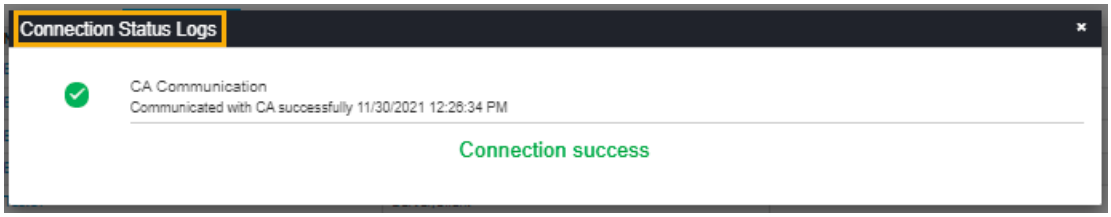
Fields	Description
***Retry Frequency**	The duration of the polling. enter the value between 1 and 30seconds
**: Mandatory fields*	

7. Click **Fetch Custom Attributes**.
   The attributes available for the CA account will be fetched from the Certificate Authority along with the CA and profile names. A pop-up message is displayed as **CA and profiles fetched**.
8. Click **Save**.
   The created Entrust configuration settings will be added. A pop-up message is displayed as **<CA_name> Settings Added**.

Validating Entrust CA

Once the Entrust settings are added, validation needs to be done to check whether the connection between AppViewX and Entrust is properly configured.

1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Certificate Authority**.
2. From the displayed CA, Select **Entrust**.
   The **Entrust** home page is displayed.
3. In the Status column of the grid with the listed accounts, click **Check** to validate the CA setting that has been created.
   The CA communication will be validated and the **Connection Status** will be shown as either **Success** or **Failure**.



Entrust MPKI

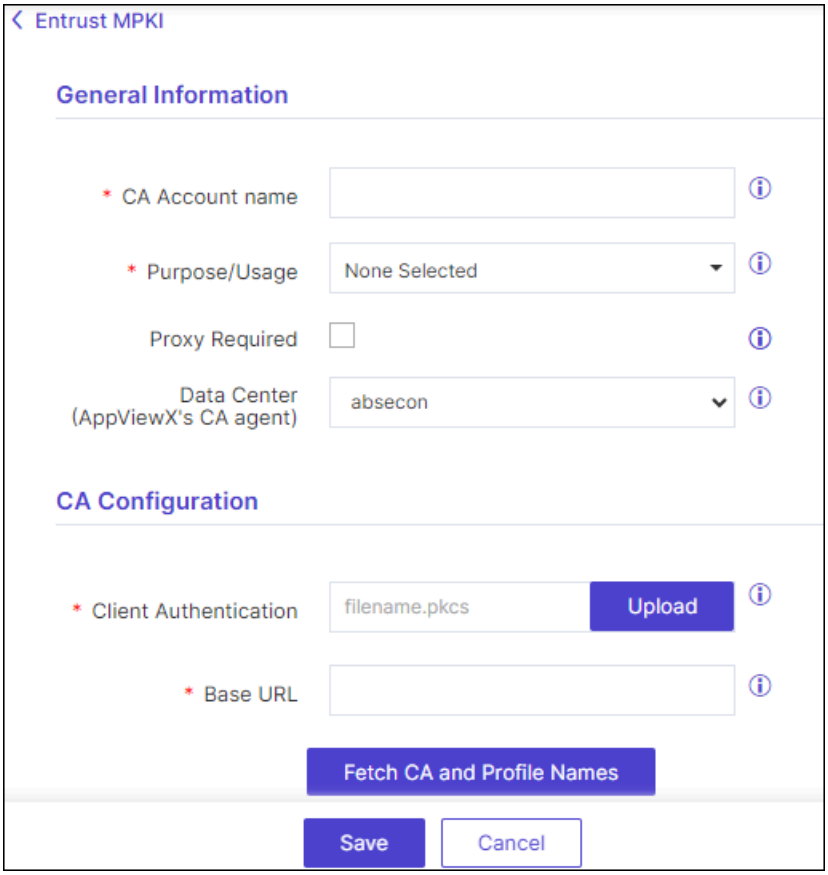
Prerequisites

The prerequisites for configuring Entrust MPKI CA account in AppViewX are as follows:

- An Entrust client authentication certificate and credentials having necessary access for CLM actions.
- AppViewX server should either have internet access or have a proxy configured in AppViewX general settings.

Configuring Entrust MPKI

1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Certificate Authority**.
2. From the displayed CA, Select **Entrust**.
   The **Entrust** home page is displayed.
3. Click the **Entrust MPKI** tab.
4. Click the **Configure Now** button or **+Add** icon from the middle or top-right of the page respectively.
   The **Entrust MPKI** configuration page is displayed.



5. Update the following details in the **General Information** section as described in the table:

General Information - Field Description Table

Fields	Description
***CA Account name**	A unique name to identify the CA setting. **Note:** No special characters other than '.', '-','_' are allowed. Names should not start with special characters.

Fields	Description
***Purpose/Usage**	Certificate Type for which CLM actions will be enabled. For example: Server and Client
Proxy Required	Enable this field if the CA communication needs to happen via Proxy. The proxy details configured in general settings will be used for communication.
Data Center (AppViewX's CA agent)	Select the data center through which the CA communication needs to happen.
*: *Mandatory fields*	

6. Update the following details in the **CA Configuration** section as described in the table. These fields are necessary for invoking the Entrust MPKI CA APIs for Certificate Management.

CA Configuration - Field Description Table

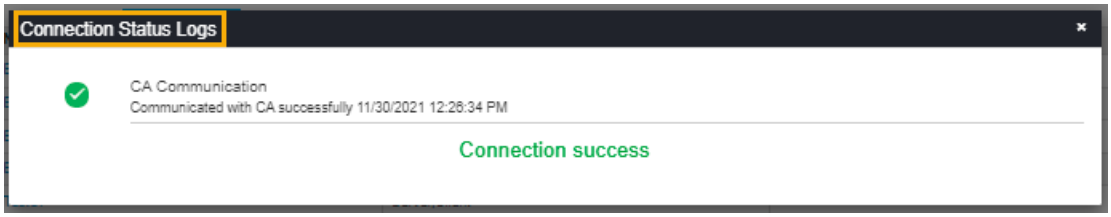
Fields	Description
***Client Authentication**	Client authentication certificate for API communication. **Note:** Must be a valid <.*p12*> file.
***Base URL**	This URL will contain just the hostname of the Entrust CA instance. Eg - https://api.entrust.net/enterprise/v2
*: *Mandatory fields*	

7. Click **Fetch CA and Profile Names.**
   The attributes available for the CA account will be fetched from the Certificate Authority along with the CA and profile names. A pop-up message is displayed as **CA and profiles fetched**.
8. Click **Save**.
   The created Entrust MPKI configuration settings will be added. A pop-up message is displayed as **<CA_name> Settings Added**.

Validating Entrust MPKI

Once the Entrust settings are added, validation needs to be done to check whether the connection between AppViewX and Entrust is properly configured.

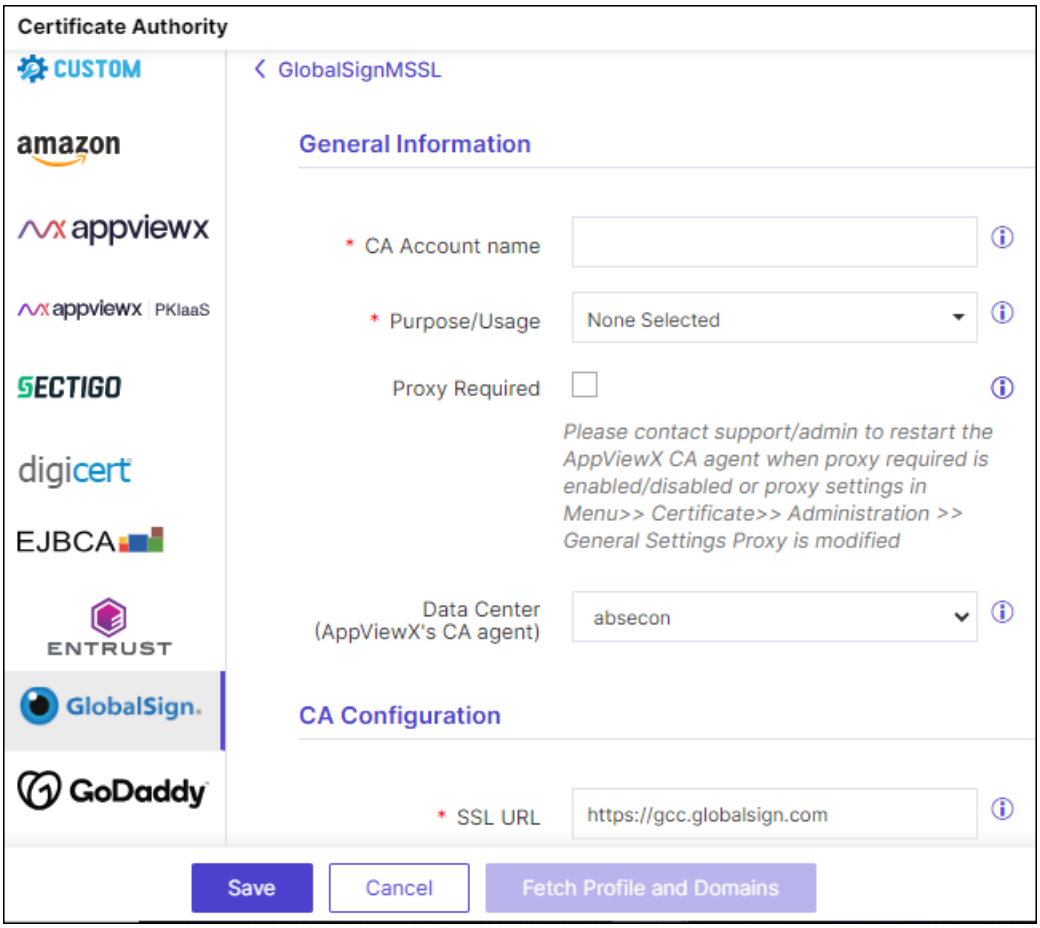
1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Certificate Authority**.
2. From the displayed CA, Select **Entrust**.
   The **Entrust** home page is displayed.
3. Click **Entrust MPKI** from the left pane of the page.
   The **Entrust MPKI** home page is displayed.
4. In the Status column of the grid with the listed accounts, click **Check** to validate the CA setting that has been created.
   The CA communication will be validated and the **Connection Status** will be shown as either **Success** or **Failure**.



GlobalSign MSSL CA

Configuring GlobalSign MSSL

1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Certificate Authority**.
2. From the displayed CA, Select **GlobalSign**.
   The **GlobalSign** home page is displayed.
3. Click the **GlobalSign MSSL** tab.
4. Click the **Configure Now** button or **+Add** icon from the middle or top-right of the page respectively.
   The **GlobalSign MSSL** configuration page is displayed.



5. Update the following details in the **General Information** section as described in the table.

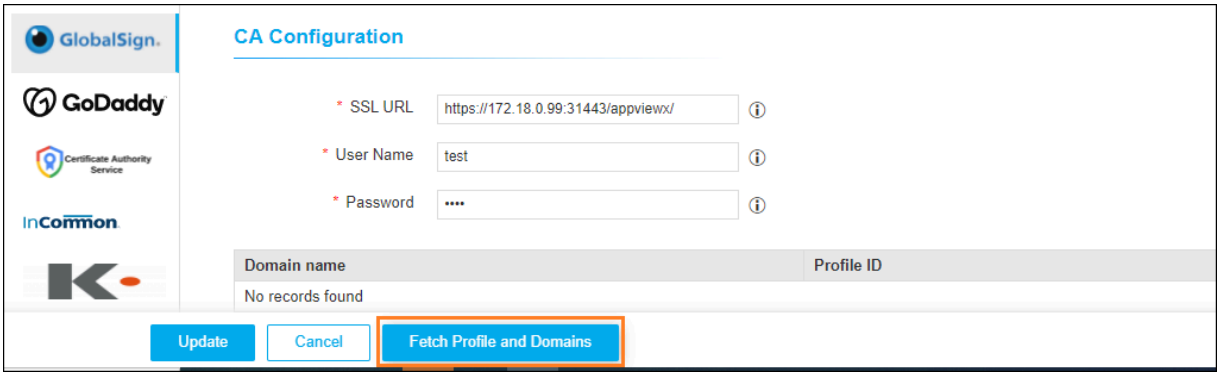
General Information - Field Description Table

Fields	Description
***CA Account name**	A unique name to identify the CA setting. No special characters other than '.', '-','_' are allowed. The name should not start with special characters.
***Purpose/Usage**	Certificate Type for which CLM actions will be enabled. For example, server and clients
Proxy Required	Enable this field if the CA communication needs to happen via Proxy. The proxy details configured in general settings will be used for communication.
Data Center (AppViewX's CA agent)	Select the data center through which the CA communication needs to happen.
*: *Mandatory fields*	

6. Update the following details in the **CA Configuration** section as described in the table.

Fields	Description
***SSL URL**	Base URL of the SSL API
***User Name**	Provide a username of the GCC to communicate with the CA.
***Password**	Provide a password for the GCC to communicate with the CA.
*: *Mandatory fields*	

7. Once all the details are configured, click **Save**.
8. In GlobalSign MSSL, we can now fetch profiles and domains by clicking on the **Fetch Profiles and Domain** button.

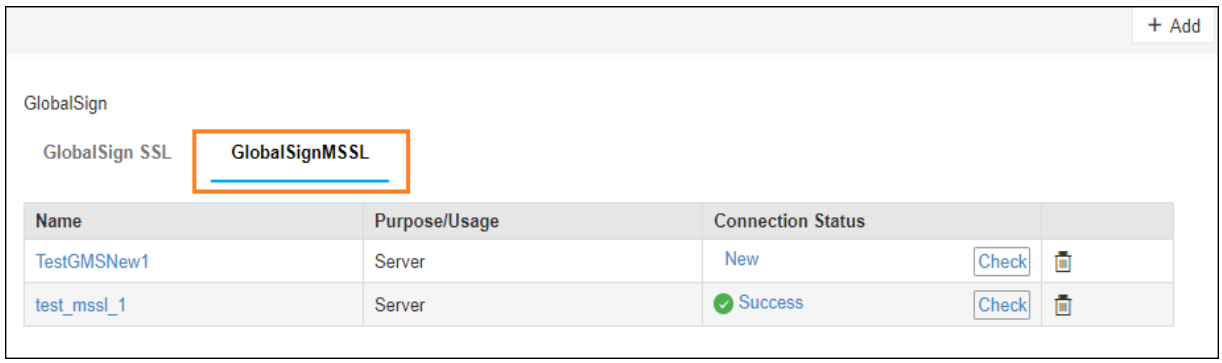


Note: The supported CSR key types are RSA **2048-8192, ECC P-256, ECC P-384** .

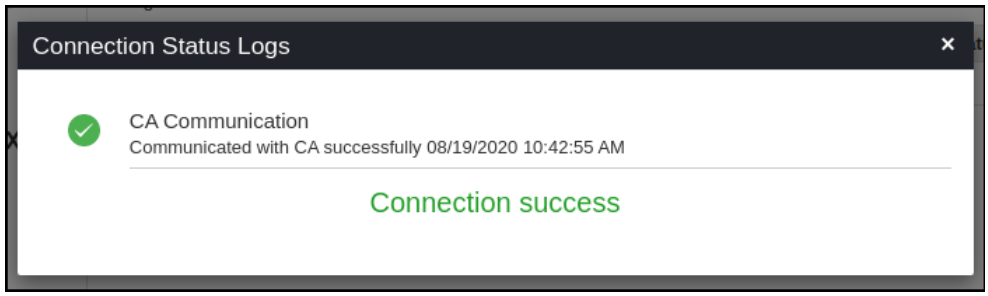
Validating GlobalSign MSSL

Once the GlobalSign MSSL settings are added, validation needs to be done to check whether the connection between AppViewX and GlobalSign MSSL is properly configured.

1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Certificate Authority**.
2. From the displayed CA, Select **GlobalSign**.
   The **GlobalSign** home page is displayed.
3. In the Status column of the grid with the listed accounts, click **GlobalSign MSSL** from the left pane of the page.
   The **GlobalSign MSSL** home page is displayed.



4. In the Status column of the grid with the listed accounts, click **Check** to validate the CA setting that is created.
5. CA communication will be validated and the Connection Status will be shown as either Success or Failure.



Limitations

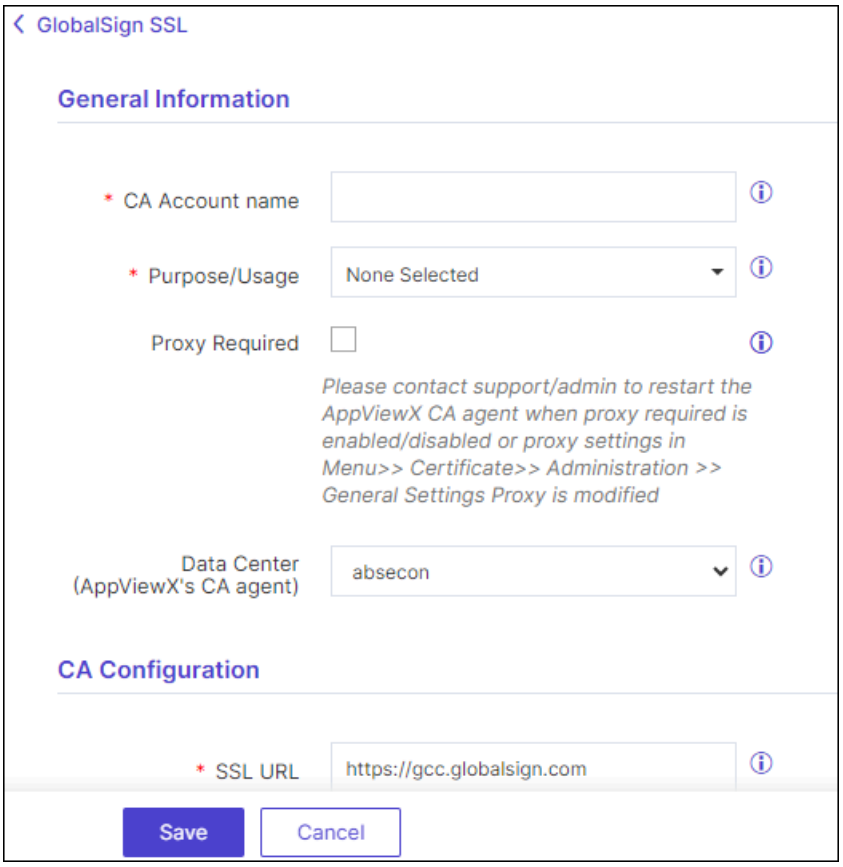
Case/ Ticket number	Fix Description
CA Setting Update	Users need to click on the **Cancel** button once the MSSL domain/profile. ID details are fetched from the GlobalSign MSSL account.

Case/ Ticket number	Fix Description
	If the user clicks the Update button, MSSL domain/profile ID details will be removed from the associated policy. The steps to follow to update CA settings are as follows: 1. On the GlobalSign MSSL CA settings page, after adding/editing values, click the Update button. 2. Navigate back to updated CA settings and click the **Fetch Profiles and Domain** button. 3. Click the **Cancel** button instead of Update to bypass the existing issue.
Default CA policy mapping	The default CA policy is defined with all available values selected and validity data is mapped based on commonly used validity. Hence, it will not have values equivalent to API documents or CA portals. This can be modified or updated in the application accordingly to the default CA policy if changes are required.
Email Address	The email address provided in the email address field on the enrollment page is not considered as the primary email value during CLM actions, instead, the email address field defined in the contact information of the logged-in user will be used. The help info message besides the Email address field on enroll/edit page is as – “If the user email address is configured, that will be used for GlobalSign CA approval actions. If the user email is not configured, then the email address provided in this field will be used" - the second part is not valid anymore.

GlobalSign SSL CA

Configuring GlobalSign SSL

1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Certificate Authority**.
2. From the displayed CA, Select **GlobalSign**.
   The **GlobalSign** home page is displayed.
3. Click the **GlobalSign SSL** tab.
4. Click the **Configure Now** button or **+Add** icon from the middle or top-right of the page respectively.
   The **GlobalSign SSL** configuration page is displayed.



5. Update the following details in the **General Information** section as described in the table.

General Information - Field Description Table

Fields	Description
***CA Account name**	A unique name to identify the CA setting. No special characters other than '.', '-','_' are allowed. The name should not start with special characters.
***Purpose/Usage**	Certificate Type for which CLM actions will be enabled. For example, Server and Client.
Proxy Required	Enable this field if the CA communication needs to happen via Proxy. The proxy details configured in general settings will be used for communication.
Data Center (AppViewX's CA agent)	Select the data center through which the CA communication needs to happen.
*: *Mandatory fields*	

6. Update the following details in the **CA Configuration** section as described in the table.

CA Configuration - Field Description Table

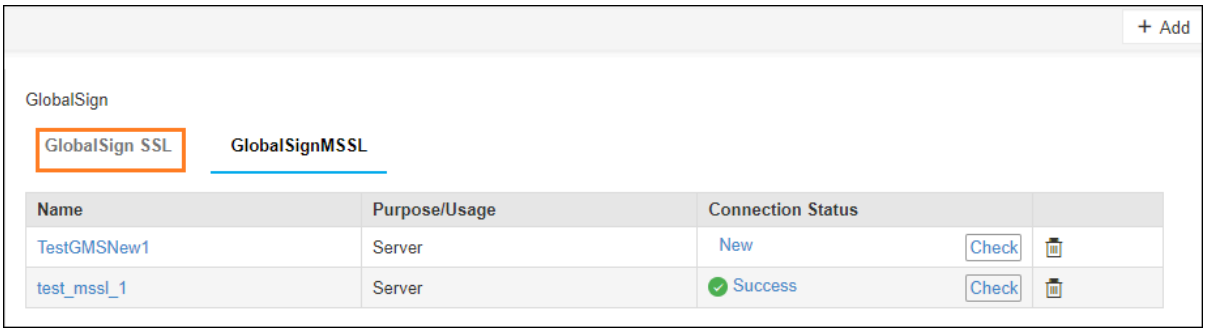
Fields	Description
***SSL URL**	Base URL of the SSL API.
***User Name**	Provide a username of the GCC to communicate with the CA.
***Password**	Provide a password for the GCC to communicate with the CA.
*: *Mandatory fields*	

7. Click **Save**.

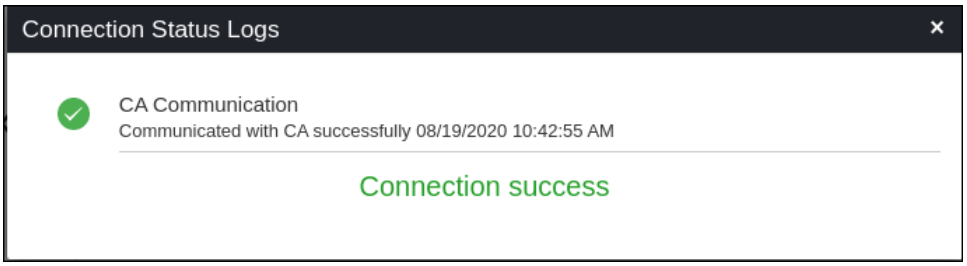
Validating GlobalSign SSL

Once the GlobalSign SSL settings are added validation needs to be done to check whether the connection between AppViewX and GlobalSign SSL is properly configured.

1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Certificate Authority**.
2. From the displayed CA, Select **GlobalSign**.
   The **GlobalSign** home page is displayed.
3. Click **GlobalSign SSL** from the left pane of the page.
   The **GlobalSign SSL** home page is displayed.



4. In the Status column of the grid with the listed accounts, click **Check** to validate the CA setting that is created.
   CA communication will be validated and the Connection Status will be shown as either Success or Failure.



GlobalSign Atlas CA

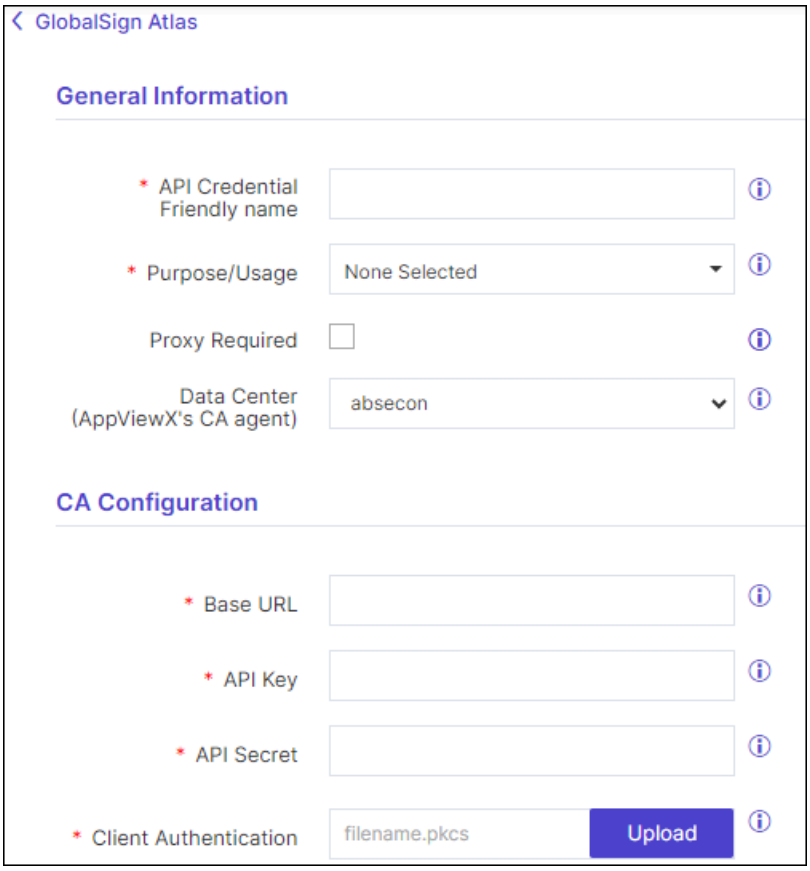
Prerequisites

The prerequisites for configuring GlobalSign Atlas CA in AppViewX are as follows:

- Login and password to access AppViewX.
- Base URL, API Key, API Secret Key.
- A client certificate provided by the GlobalSign Atlas team.

Configuring GlobalSign Atlas

1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Certificate Authority**.
2. From the displayed CA, Select **GlobalSign**.
   The **GlobalSign** home page is displayed.
3. Choose the **GlobalSign Atlas** tab.
4. Click the **Configure Now** button or **+Add** icon from the middle or top-right of the page respectively.
   The **GlobalSign Atlas** configuration page is displayed.



5. Update the following details in the **General Information** section as described in the table.

General Information - Field Description Table

Fields	Description
***API Credential Friendly name**	Enter the API Credentials Friendly name (which is the CA Account name that will be used for the CA Policy and Enrollment).
***Purpose/Usage**	Select the purpose of the certificate that can be requested using this account. **Note:** Users can select *Server*, *Client* or both.
Proxy Required	Select the checkbox if communication to the Certificate Authority (CA) has to use the proxy details provided in the general settings
Data Center (AppViewX's CA agent)	Select the data center that will be used to establish the communication with the CA.

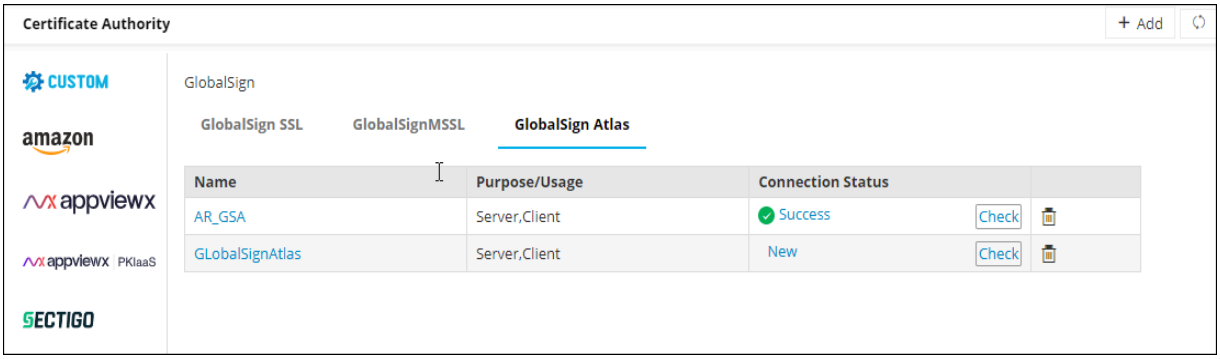
Fields	Description
*: *Mandatory fields*	

6. Update the following details in the **CA Configuration** section as described in the table.

CA Configuration - Field Description Table

Fields	Description
***Base URL**	Enter the base URL required for constructing the API request.
***API Key**	Enter the API key which is the unique identifier used to authenticate a user. **Note:** The API Key will be displayed as asterisks (*)
***API Secret**	Enter the API secret to communicate with the CA. **Note:** The API Key will be displayed as asterisks (*)
***Client Authentication**	Upload the certificate for client authentication in the .p12 or .pfx format only.
*: *Mandatory fields*	

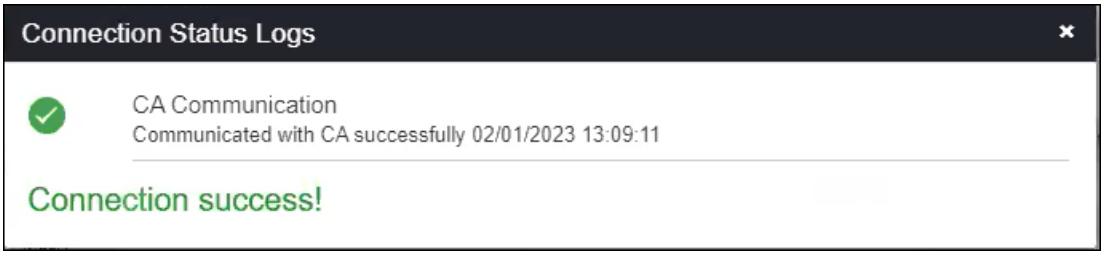
7. Click the **Fetch Validation Policy and Save** button.
   A confirmation message will appear “Validation Policy fetched and settings have been updated.” and the CA is created successfully. The connection status for the CA is displayed as New.



Validating GlobalSign Atlas

Once the GlobalSign Atlas settings are added, validation needs to be done to check whether the connection between AppViewX and GlobalSign Atlas is properly configured.

1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Certificate Authority**.
2. From the displayed CA, Select **GlobalSign**.
   The **GlobalSign** home page is displayed.
3. Click **GlobalSign Atlas** from the left pane of the page.
   The **GlobalSign Atlas** home page is displayed.
4. In the Status column of the grid with the listed accounts, click **Check** to validate the CA setting that is created.
5. CA communication will be validated and the Connection Status will be shown as either Success or Failure.



GoDaddy CA

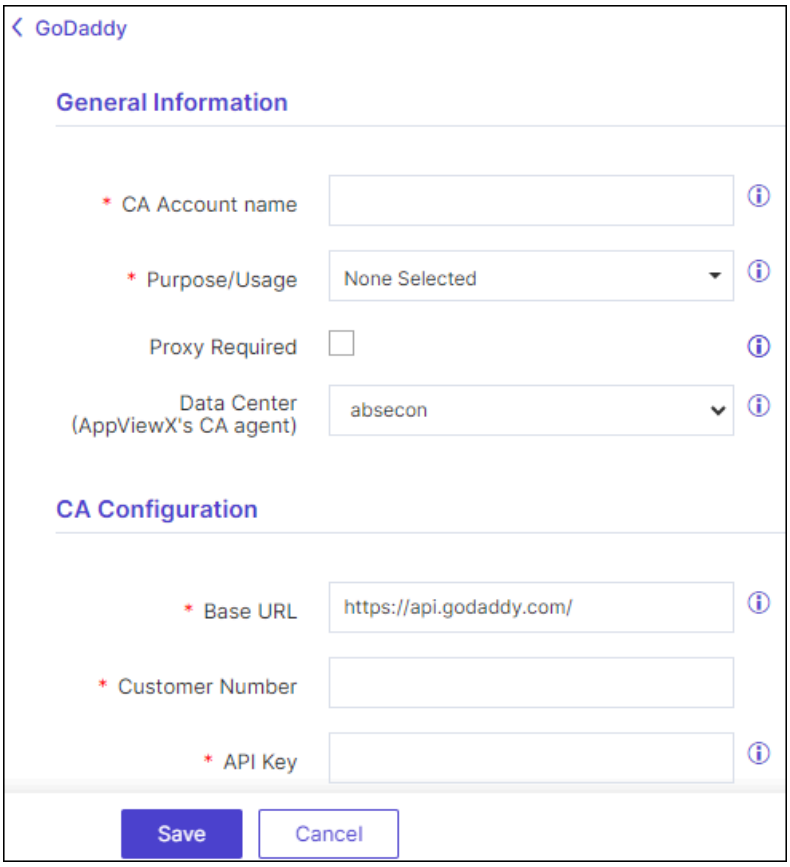
Prerequisites

The prerequisites for configuring GoDaddy CA in AppViewX are as follows:

1. GoDaddy Customer Number, API key, and secret are required to make API Requests from AppViewX in order to perform CLM (Certificate Lifecycle Management) operations.
2. AppViewX server should either have internet access or have a proxy configured in AppViewX general settings.
3. Customer Number, API key, and secret configuration in GoDaddy Account:
   a. After logging into the GoDaddy portal with proper account credentials go to https://developer.godaddy.com/keys
   b. Users will be asked to add an optional name, and the secret will be displayed which needs to be copied and will not be displayed further.
   c. This API key and secret will be used for further communication.
   d. Customer Number details are available on the Accounts page of the GoDaddy website
4. Product units should be available in the customer's GoDaddy account to perform CLM operations.

Configuring GoDaddy

1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Certificate Authority**.
2. From the displayed CA, Select **GoDaddy**.
   The **GoDaddy** home page is displayed.
3. Click the **Configure Now** button or **+Add** icon from the middle or top-right of the page respectively.
   The **GoDaddy** configuration page is displayed.



4. Update the following details in the **General Information** section as described in the table:

General Information - Field Description Table

Fields	Description
***CA Account name**	A unique name to identify the CA setting. **Note:** No special characters other than '.', '-','_' are allowed. Names should not start with special characters.
***Purpose/Usage**	Certificate Type for which CLM actions will be enabled.

Fields	Description
	Example: Server, Client.
Proxy Required	Enable this field if the CA communication needs to happen via Proxy. The proxy details configured in general settings will be used for communication.
Data Center (AppViewX's CA agent)	Select the data center through which the CA communication needs to happen.
*: *Mandatory fields*	

5. Update the following details in the **CA Configuration** section as described in the table. These fields are necessary for invoking the GoDaddy CA APIs for Certificate Management.

CA Configuration - Field Description Table

Fields	Description
***Base URL**	This URL will contain the Base URL of the GoDaddy CA API instance. For example: https://api.godaddy.com
***Customer Number**	Each user will have a unique customer number which is used to obtain the certificates from the GoDaddy CA account.
***API Key**	API key generated in the GoDaddy portal which is used for GoDaddy API communications.
***API Secret**	API Secret generated in the GoDaddy portal which is used for GoDaddy API communications.
First Name	First name of the GoDaddy Account user’s name as provided in the portal to be used for certificate creation purposes.
Last Name	Last name of the GoDaddy Account user’s name as provided in the portal to be used for certificate creation purposes.
Email Address	Email Id of the GoDaddy Account user’s name as provided in the portal to be used for certificate creation purposes. **Note**: Valid email address.

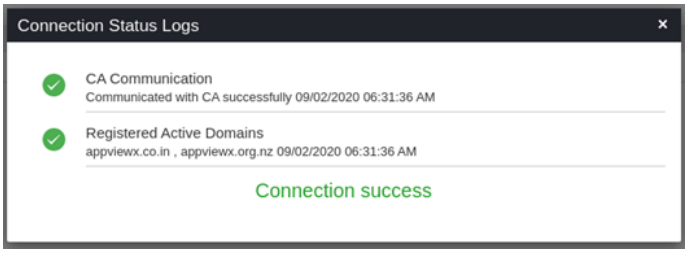
Fields	Description
Phone Number	Phone number of the GoDaddy Account user as provided in the portal to be used for certificate creation purposes. **Note:** Phone numbers must contain a minimum of 7 and a maximum of 15 numeric values.
*: *Mandatory fields*	

6. Click **Save**.

Validating GoDaddy

Once the GoDaddy settings are added validation needs to be done to check whether the connection between AppViewX and GoDaddy is properly configured.

1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Certificate Authority**.
2. From the displayed CA, Select **GoDaddy**.
   The **GoDaddy** home page is displayed.
3. In the Status column of the grid with the listed accounts, click **Check** to validate the CA setting that has been created.
   The CA communication will be validated and the **Connection Status** will be shown as either **Success** or **Failure**.

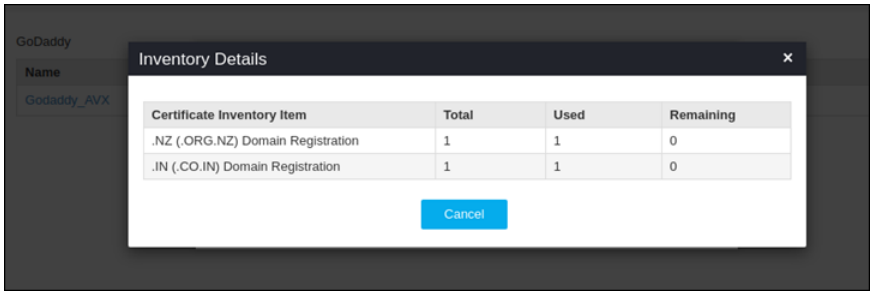


View GoDaddy Product Units

Each GoDaddy account has different types of SSL products and units. The below steps will allow users to know the availability of the products and their remaining units.

1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Certificate Authority**.
2. From the displayed CA, Select **GoDaddy**.
   The **GoDaddy** home page is displayed.

3. On the **GoDaddy** page, click **View** to fetch the product types and the units available for the GoDaddy account configured.
   Once clicked, users can view the available and used product units.



Google CA

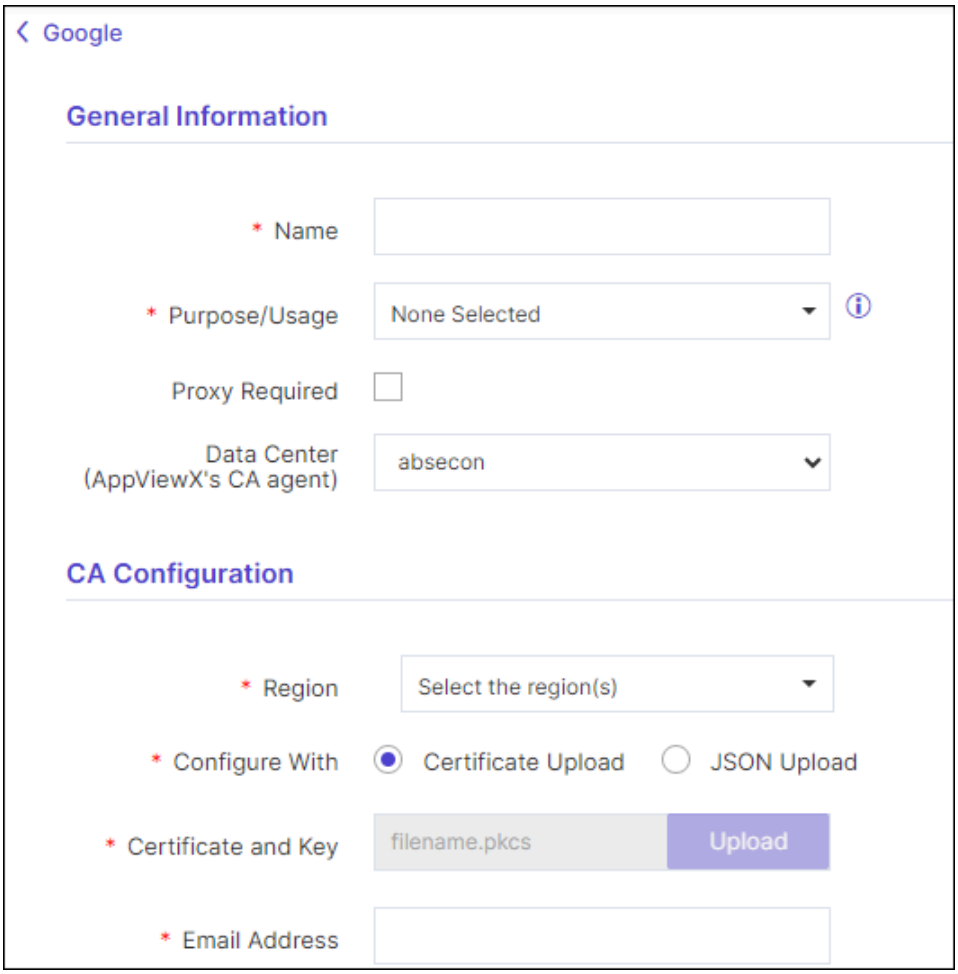
Prerequisites

The prerequisites for configuring a Google CA account in AppViewX are as follows:

- A Google client certificate or Google client authentication Json for a user having necessary access for enrolling the certificates and for other Certificate Lifecycle Management(CLM) operations.
- AppViewX servers should either have internet access or have a proxy configured in AppViewX general settings.
- The URL https://www.googleapis.com should be reachable from AppViewX.

Configuring Google

1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Certificate Authority**.
2. From the displayed CA, Select **Google**.
   The **Google** home page is displayed.
3. Click the **Configure Now** button or **+Add** icon from the middle or top-right of the page respectively.
   The **Google** configuration page is displayed.



4. Update the following details in the **General Information** section as described in the table:

General Information - Field Description Table

Fields	Description
***CA Account name**	A unique name to identify the CA setting. **Note:** No special characters other than '.', '-','_' are allowed. The name should not start with special characters.
***Purpose/Usage**	Certificate Type for which CLM actions will be enabled. For example, Server and Client
Proxy Required	Enable this field if the CA communication needs to happen via Proxy. The proxy details configured in general settings will be used for communication.

Fields	Description
Data Center (AppViewX's CA agent)	Select the data center through which the CA communication needs to happen.
*: *Mandatory fields*	

5. Configure either Certificate Upload or JSON Upload. These fields are necessary for invoking the Google CA APIs via Certificate Upload for Certificate Management. Select the Certificate Upload check box,
   Update the following details in the **CA Configuration** section as described in the table.

Fields	Description
***Certificate and Key**	Client authentication certificate for API communication.
***Email address**	Email address of the user
***Project Id**	Id of the project
*: *Mandatory fields*	

6. Select the JSON Upload check box and configure a CA. Click the Upload button to upload the JSON file.
7. Click **Validate and Fetch**. The issuer names available for the CA account will be fetched along with the validity of the issuers from the Certificate Authority.

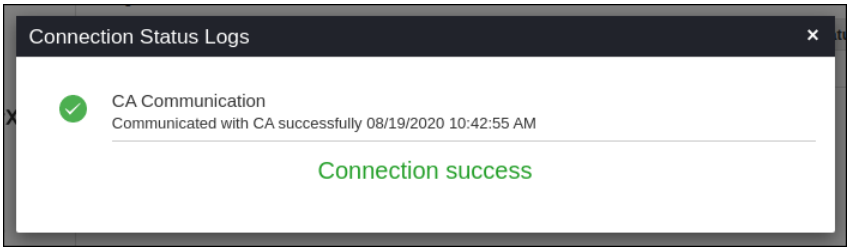
Location	CA Name	Validity	Delete
us-central1	pre-prod-root-ca	05/13/2030 20:02:21	
	testbed-root-ca	05/13/2030 20:27:49	
	prod-root-ca	04/23/2030 12:23:32	
	prod-inter-ca-level-981	06/14/2020 09:03:33	
	prod-inter-ca-level-200	06/14/2020 09:08:43	
	prod-inter-ca-level-201	06/14/2020 09:09:09	
	prod-inter-ca-level-000	06/14/2020 08:51:09	
	prod-inter-ca	04/23/2030 12:27:40	
	prod-inter-ca-level-01	06/14/2020 09:00:50	
europe-west1	test-bed-root-ca	05/13/2030 21:09:13	

8. Click **Save**.

Validating Google

Once the Google settings are added validation needs to be done to check whether the connection between AppViewX and Google is properly configured.

1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Certificate Authority**.
2. From the displayed CA, Select **Google**.
   The **Google** home page is displayed.
3. In the Status column of the grid with the listed accounts, click **Check** to validate the CA setting that is created.
   CA communication will be validated and the Connection Status will be shown as either Success or Failure.



HashiCorp Vault CA

Prerequisites

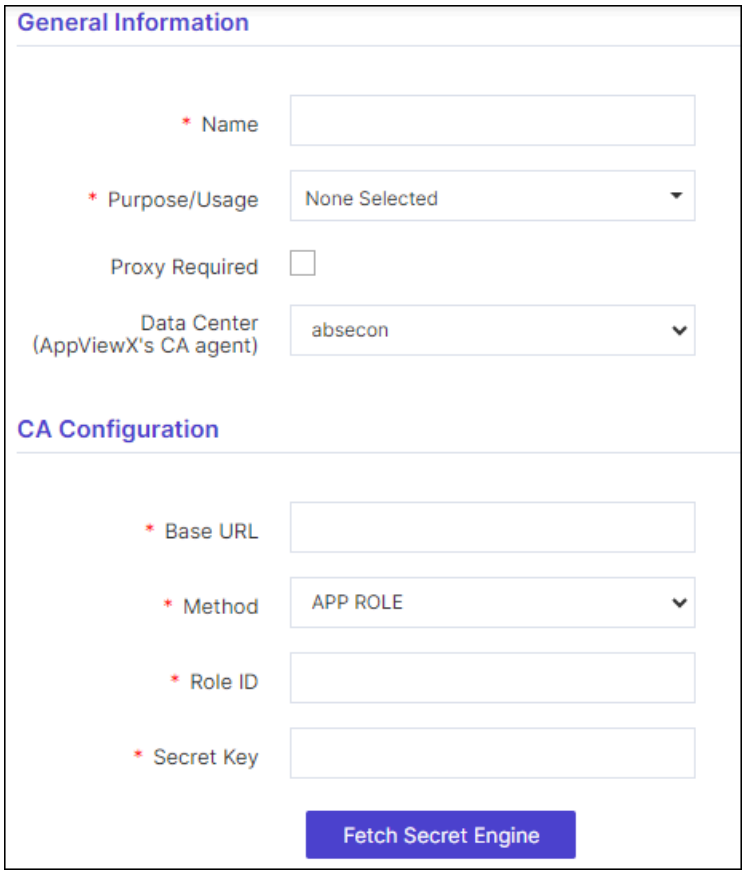
The prerequisites for configuring Hashicorp Vault CA account in AppViewX are as follows:

- Login and password to access AppViewX.
- Base URL, Role ID, and Secret Key for the **APP ROLE method** and Base URL, Access Key, Secret Key, and Role Name for the **AWS method** in the CA Configurations.

Configuring HashiCorp Vault

Steps to Configure HashiCorp Vault CA

1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Certificate Authority**.
2. From the displayed CA, Select **HashiCorp Vault**.
   The **HashiCorp Vault** home page is displayed.
3. Click the **Configure Now** button or **+Add** icon from the middle or top-right of the page respectively.
   The **HashiCorp Vault** configuration page is displayed.



4. Update the details in the General Information section as described in the table below:

General Information - Field Description Table

Fields	Description
***CA Account name**	A unique name to identify the CA setting. **Note:** No special characters other than '.', '-', and '_' are allowed. Names should not start with special characters.
***Purpose/Usage**	The dropdown contains checkboxes for the certificate type for which the CLM actions will be enabled. The values are: • Server • Client • Code Signing

Fields	Description
	One or more values can be selected depending on the type of account users need to create.
Proxy Required	Enable this field if the CA communication needs to happen via Proxy. The proxy details configured in general settings will be used for communication.
Data Center (AppViewX's CA agent)	Select the data center through which the CA communication needs to happen.
*: *Mandatory fields*	

5. Update the details in the **CA Configuration** section as described in the tables below. These fields are necessary for invoking the Hashicorp CA Secret Engine for Certificate Management. The fields displayed in the CA Configuration section depend on the value selected in the **Method** field. The two auth method values are
   - **APP ROLE** - The APP ROLE auth method allows machines or apps to authenticate with Vault-defined roles. An "AppRole" represents a set of Vault policies and login constraints that must be met to receive a token with those policies. An AppRole can be created for a particular machine, or even a particular user on that machine or a service spread across machines. The credentials required for successful login depend upon the constraints set on the AppRole associated with the credentials.
   - **AWS** - The AWS auth method provides an automated mechanism to retrieve a Vault token for IAM principals and AWS EC2 instances. Unlike most Vault auth methods, this method does not require manual first-deploying or provisioning of security-sensitive credentials (tokens, username/password, client certificates, etc), by operators under many circumstances.

CA Configuration for App Role - Field Description Table

Fields	Description
***Base URL**	The base of URL of the CA account.
***Method**	**APP ROLE**
***Role ID**	RoleID is an identifier that selects the AppRole against which the other credentials are evaluated. When authenticating against this auth method's login endpoint, the RoleID is a required argument at all times. By default, RoleIDs are unique UUIDs, which allow them to serve as secondary secrets to the other credential information.

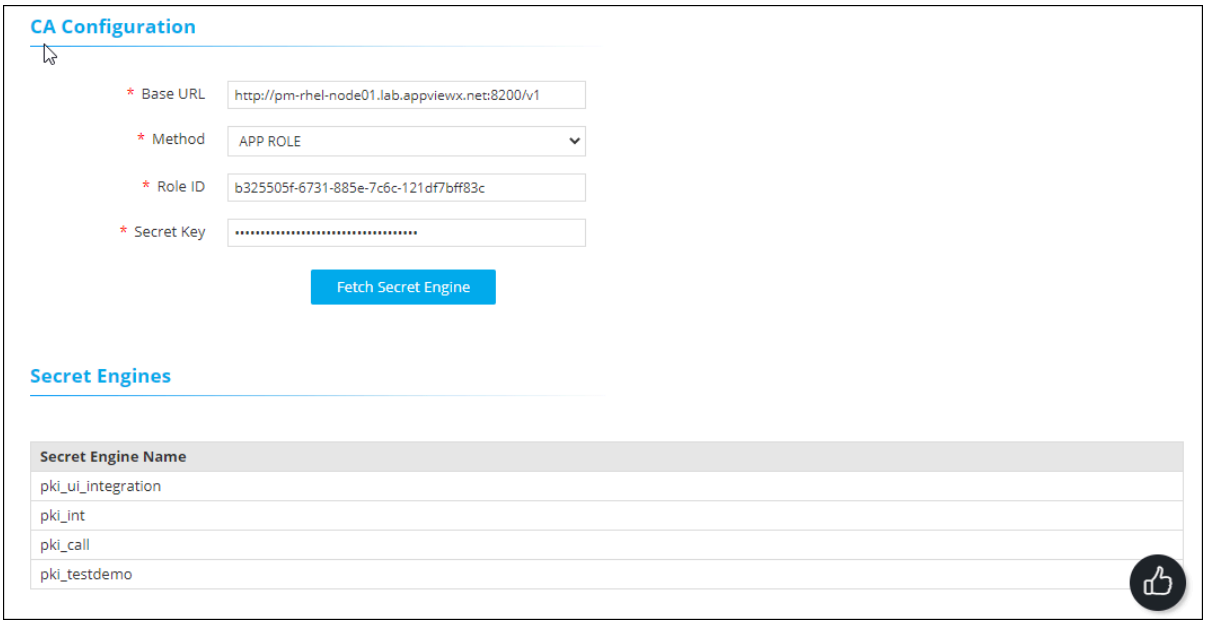
Fields	Description
***Secret Key**	Secrey Key (SecretID) is a credential that is required by default for any login and is intended to always be secret. They can be created against an AppRole either via generation of a 128-bit purely random UUID by the role itself or via specific, custom values. Similarly to tokens, Secret keys have properties like usage-limit, TTLs and expirations.
*: *Mandatory fields*	

CA Configuration for AWS - Field Description Table

Fields	Description
***Base URL**	The base of URL of the CA account.
***Method**	**AWS**
***Access Key**	Access Keys are used to sign the requests that are sent. Access Key and Secret Key are used for programmatic (API) access to AWS services.
***Secret Key**	Secret Key (SecretID) is a credential that is required by default for any login and is intended to always be secret. Similar to tokens, SecretIDs have properties like usage-limit, TTLs, and expirations.
***Role Name**	The basic mechanism of operation (AWS authorization workflow) is per-role. Roles are registered in the method and associated with a specific authentication type that cannot be changed once the role has been created. Roles can also be associated with various optional restrictions, such as the set of allowed policies and max TTLs on the generated tokens.
*: *Mandatory fields*	

The correct values entered in the fields establish a connection with the Hashicorp vault to be able to fetch the secret engine.

6. Click the **Fetch Secret Engine** button.
   A list of PKI secret engines is displayed. These will be presented to users in the policy. from where they can select the respective secret engines.



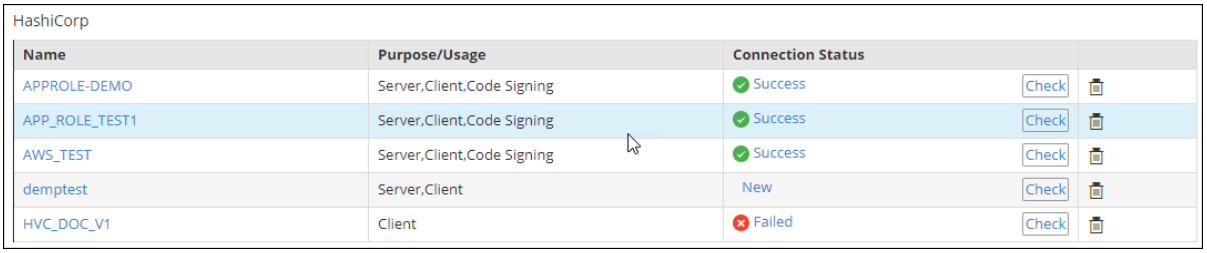
7. Click **Save**.

   The Account details are displayed in a grid at the bottom of the screen, with options to options to edit, check (connection status), and delete.

Editing an Account

1. Go to the **HashiCorp Vault** CA account home page

   The list of Accounts is displayed in the grid.



2. Click the account name from the ‘Name’ column in the grid.

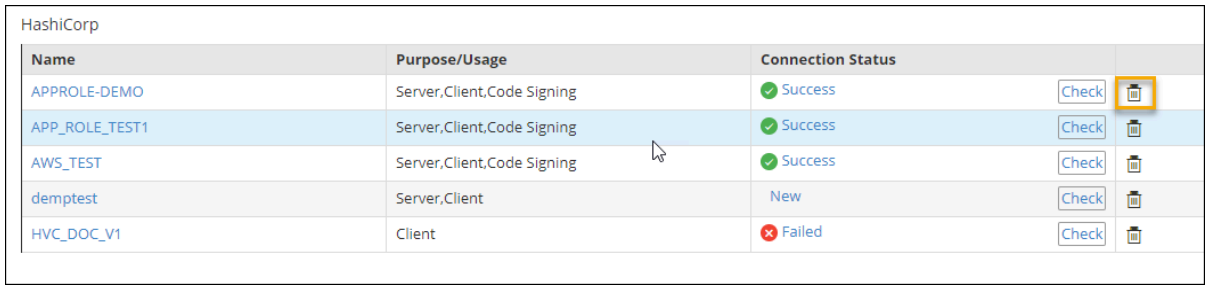
   The General Information and CA Configuration sections are displayed with pre-populated values.

3. Change any of the editable fields and click the **Fetch Secret Engine** button.
4. Click the **Update** button.

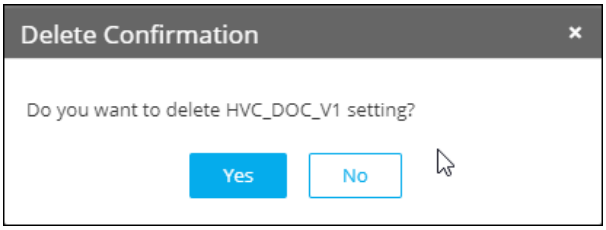
Deleting an Account

1. Go to the **HashiCorp Vault** CA account home page

   The list of Accounts is displayed in the grid.



2. In the last column of the grid with the listed accounts, click the **Delete** or bin icon.
   The Delete Confirmation pop-up is displayed.



3. Click **Yes**.

Validating HashiCorp Vault

To check the connection status of an account,

1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Certificate Authority**.
2. From the displayed CA, Select **HashiCorp Vault**.
   The **HashiCorp Vault** home page is displayed.
3. In the Status column of the grid with the listed accounts, click the **Check** button.
   The Success or Failure value is displayed.
4. Update the account details accordingly to get a “success” status.

HydrantID CA

Prerequisites

The prerequisites for configuring a HydrantID CA account in AppViewX as follows:

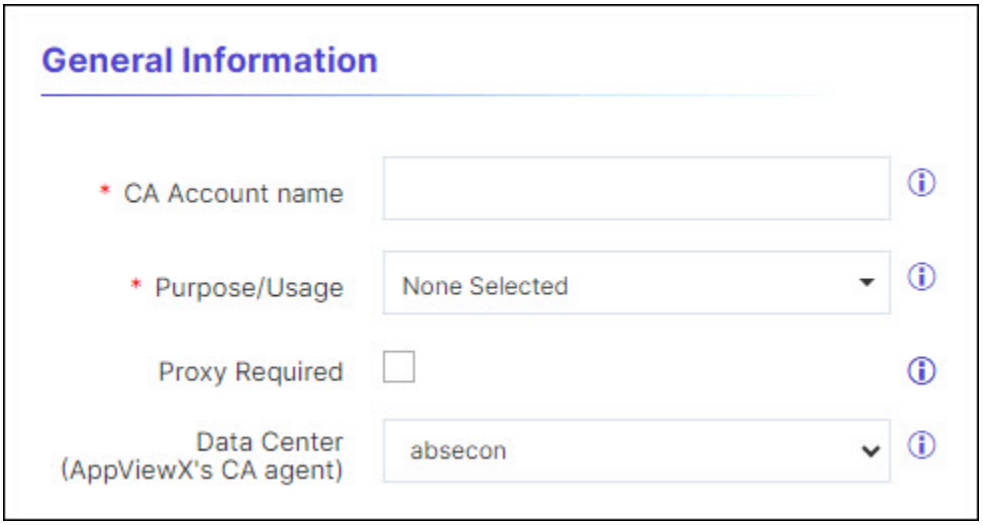
1. To create a CA configuration the following values are required:
   - Base URL
   - API Key ID
   - API Key

Once the organization (AppViewX) has subscribed for a HydrantID account, you will be provided with a **Username**, **Password**, and **Login URL**.

2. The API Key ID and API Key should be of the following User Roles in HydrantId:
   - Account Auditor
   - Organization Admin
   - Organization Auditor
   - Requestor
3. Users with role **Account Admin** in the HydrantID application can create the above roles. Only account admins can generate the API Key ID and API Key for the roles. Both values can be viewed for a limited time only. Ensure to note these values after the roles are added.

Configuring HydrantID CA

1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Certificate Authority**.
2. From the displayed CA, select **HydrantID**.
   The **HydrantID** home page is displayed.
3. Click the **Configure Now** button or **+Add** icon from the middle or top-right of the page respectively.
   The HydrantID CA details page is displayed.
4. Update the following details in the **General Information** section as described in the table:

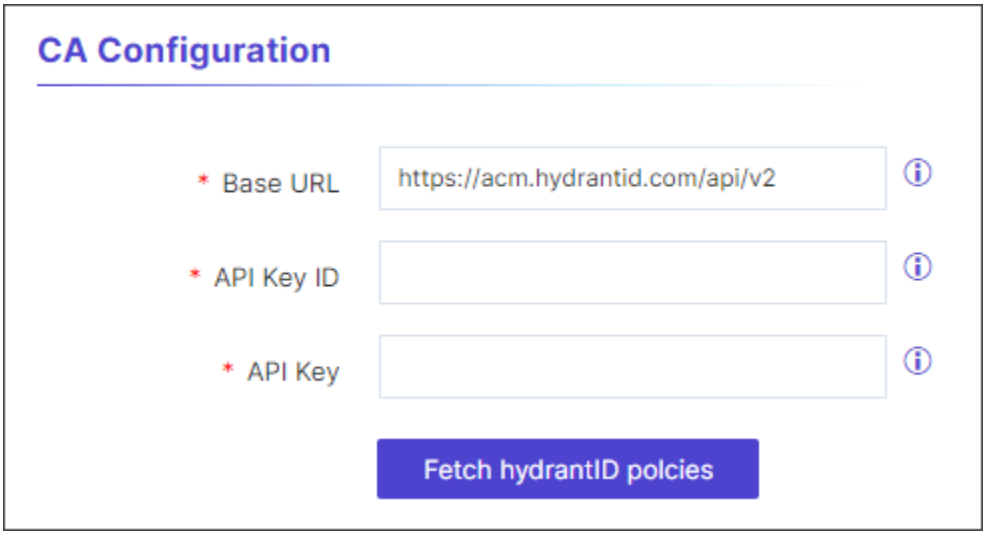


General Information - Field Description Table

Fields	Description
***CA Account name**	A unique name to identify the CA setting.

Fields	Description
	Permissible special characters are '.', '-','_'. Names should not start with special characters.
***Purpose/Usage**	Certificate Type for which CLM actions will be enabled. Server, Client and Code-signing are the supported types.
Proxy Required	Enable this field if the CA communication needs to happen via Proxy. The proxy details configured in general settings will be used for communication.
Data Center (AppViewX's CA agent)	Select the data center through which the CA communication needs to happen.
*: *Mandatory fields*	

5. Update the following details in the **CA Configuration** section as described in the table. These fields are necessary for invoking the HydrantId APIs for Certificate Management.



CA Configuration - Field Description Table

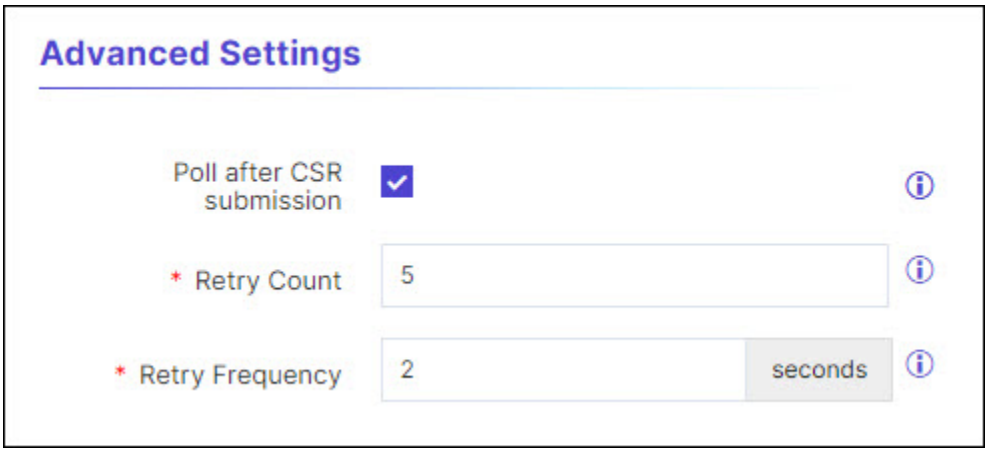
Fields	Description
***Base URL**	This URL will contain the hostname of the HydrantID CA instance and used for constructing the API requests. the default value is https://acm.hydrantid.com/api/v2

Fields	Description
***API Key ID**	Enter the API Key ID generated in the HydrantID application. Its is a unique value specific to the user created in hydrant and is used to authenticate the user.
***API Key**	Enter the API Key generated in the HydrantID application. Its is a unique value specific to the user created in hydrant and used to authenticate and authorize requests.
*: *Mandatory fields*	

6. Click **Fetch hydrantID policies**.
   A list of policies associated with the account are displayed. These are made available from HydrantID and are used for requesting different types of certificates.

> **Note:** Configuration can only be saved in AppViewX if the profiles are available.

7. Update the following details in the **Advanced Settings** section as described in the table.



Advanced Settings - Field Description Table

Fields	Description
Poll after CSR Submission	A check box field when selected will fetch the certificated immediately after CSR Submission on enrollment, renew, and reissue of certificate with the retry count and retry frequency as described below.
***Retry Count**	The number of times the polling will take place after CSR submission. Enter a value between 1 and 10.
***Retry Frequency**	The duration of the polling. enter the value between 1 and 30seconds.

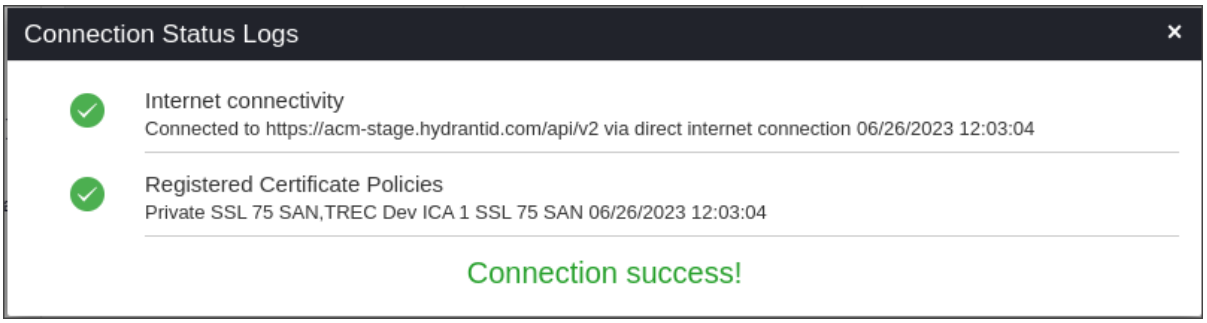
Fields	Description
*: *Mandatory fields*	

8. Click **Save**.
   The created HydrantID configuration settings will be added. A pop-up message is displayed as **<CA_name> Settings Added**.

Validating HydrantID CA

Once the HydrantID settings are added, validation needs to be done to check whether the connection between AppViewX and HydrantID is properly configured.

1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Certificate Authority**.
2. From the displayed CA, select **HydrantID**.
   The **HydrantID** home page is displayed.
3. In the Status column of the grid with the listed accounts, click **Check** to validate the CA setting that has been created.
   The CA communication will be validated and the **Connection Status** will be shown as either **Success** or **Failure**



IDnomic CA

Prerequisites

The prerequisites for configuring IDnomic CA account in AppViewX are as follows:

- **CA Base URL** - is shared by IDonomic to the users by email/shared file location or refer attached API documentation.
- **Partition name** - is shared by IDonomic to the users by email/shared file location
- **Client authentication certificate** *(.p12 or .pfx format)* - is shared by IDonomic to the users by email/ shared file location

- (*Optional*) **SOAP signing authentication certificate** *(.p12 or .pfx format)* - is shared by IDonomic to the users by email/shared file location
- (*Optional*) **RA Base URL** - is shared by IDonomic to the users by email/shared file location or refer attached API documentation.
- (*Optional*) **RA client authentication certificate** *(.p12 or .pfx format)* - is shared by IDonomic to the users by email/shared file location
- AppViewX server should either have internet access or have a proxy configured in AppViewX general settings. Refer to the section Managing Proxy Settings in the Platform guides.

Note: In the CA configuration page, if the checkbox **Use same certificate for signing SOAP requests** is selected, then **Client authentication certificate** is used as the **SOAP signing authentication certificate** and also the **RA client authentication certificate**.

Configuring IDnomic CA

1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Certificate Authority**.
2. From the displayed CA, Select **IDnomic**.
   The **IDnomic** home page is displayed.
3. Click the **Configure Now** or **+Add** icon from the middle or top-right of the page respectively.
   The IDnomic configuration page is displayed.
4. Update the following details in the **General Information** section as described in the table.

General Information - Field Description table

Fields	Description
***CA Account name**	A unique name to identify the CA setting No special characters other than '.', '-','_' are allowed. Names should not start with special characters.
***Purpose/Usage**	Certificate Type for which CLM actions will be enabled. Example: Server, Client.
Proxy Required	Enable this field if the CA communication needs to happen via Proxy. The proxy details configured in general settings will be used for communication.

Fields	Description
Data Center (AppViewX's CA agent)	Select the data center through which the CA communication needs to happen.
**: Mandatory fields*	

5. Update the following details in the **CA Configuration** section as described in the table below.

Figure 1. Default CA Configuration

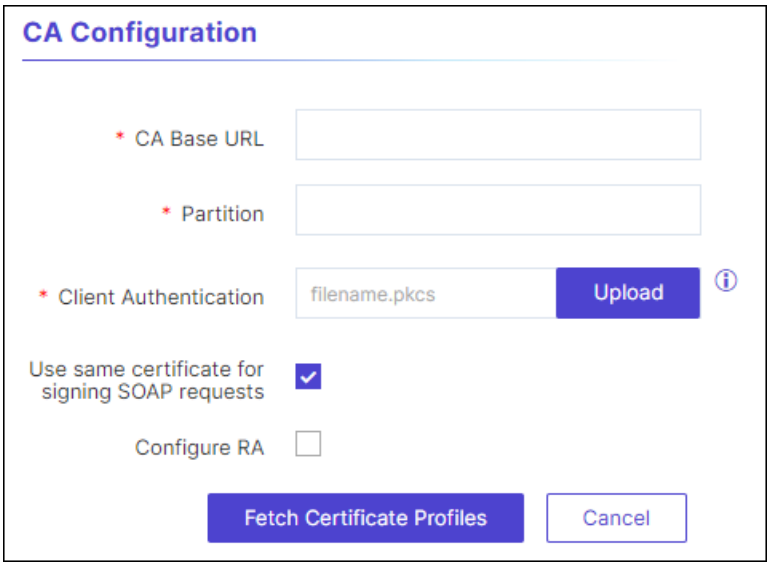
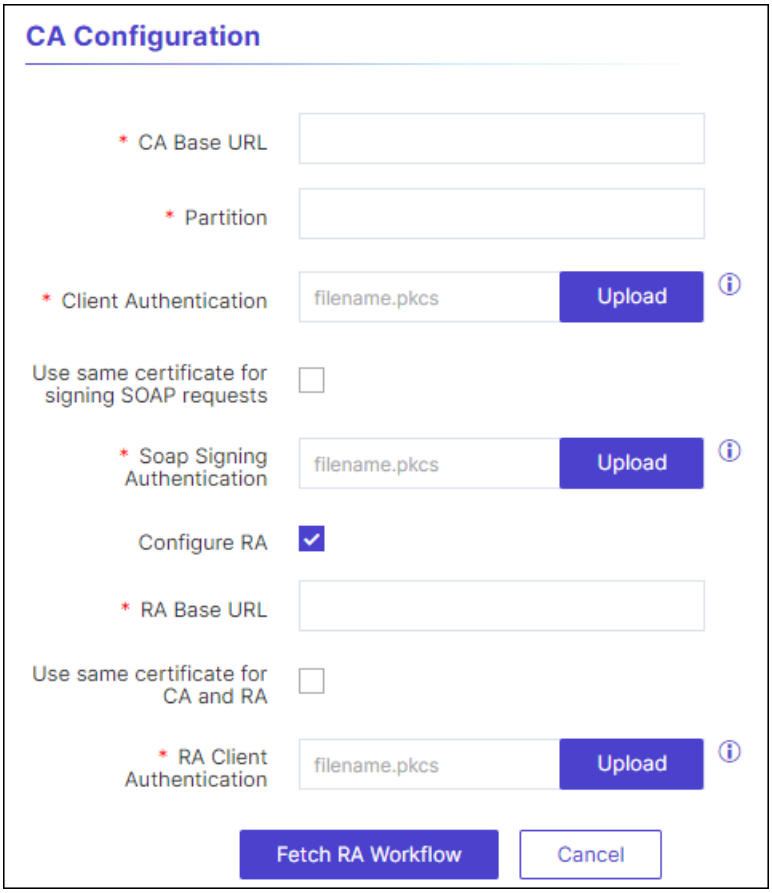


Figure 2. CA Configuration with RA



CA Configuration - Field Description table

Fields	Description
***CA Base URL**	Enter the base URL of the IDnomic CA API instance. For example: https://api-ca.idnomic.com
***Partition**	The name of the partition assigned to the organization's application; it is the workspace dedicated to the organization. The partitioning system allows implementing multi-tenancy within ID CA and ID RA. Partitions are defined in a tree structure, each node being a partitioning in which resources (e.g. Configurations, Certificate, etc) can be attached. According to the customer's needs we can isolate or share common resources.

Fields	Description
***Client Authentication**	Upload the certificate for client authentication in the .p12 or .pfx format only.
Use same certificates for signing SOAP requests	If checked, allow user to use the same certificate uploaded in Client certificate field, if not checked, you can upload another certificate in the enabled field **Soap Signing certificate**.
***SOAP Signing Authentication**	Upload the certificate for soap signing authentication in the .p12 or .pfx format only.
Configure RA	The field is unchecked by default. If you want to allow a user or device to request a digital certificate from a specific website or application you can select the checkbox and update the fields below. A registration authority (RA) is an authority in a network that verifies user requests for a digital certificate and tells the certificate authority (CA) to issue it.
***RA Base URL**	Enter the base URL of the IDnomic RA API instance. For example: https://api-ra.idnomic.com
Use same certificate for CA and RA	This field is checked by default. In that case, use the same client certificate uploaded in the “Client Certificate” section. If unchecked, you may upload a new certificate in the enabled field labeled as **RA Client Authentication**.
***RA Client Authentication**	Upload the certificate for RA client authentication in the .p12 or .pfx format only.
**: Mandatory fields*	

6. Click **Fetch Certificate Profiles**. (If RA is used the button label changes to **Fetch RA Workflow**)
   If only CAs are used in the configuration, then a list of certificate profiles are displayed and if RAs are configured then only the certificate RA workflows are displayed.

7. Click **Save**.
   A confirmation message will appear “Validation Policy fetched and settings have been updated.” and the CA is created successfully. The connection status for the CA is displayed as New.

Validating IDnomic CA

Once the IDnomic settings are added validation needs to be done to check whether the connection between AppViewX and IDnomic is properly configured. To validate the IDnomic CA,

1. Go to **(Menu) > SIGN+ > ADMINISTRATION > Certificate Authority.**
2. From the displayed CA, Select **IDnomic**.
3. In the Status column of the grid with the listed accounts, click **Check** to validate the CA setting that is created.
   The CA communication will be validated and the Connection Status will be shown as either Success or Failure.

Figure 3. RA Validation

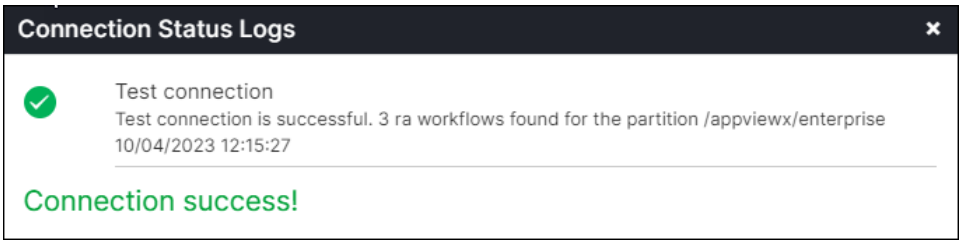
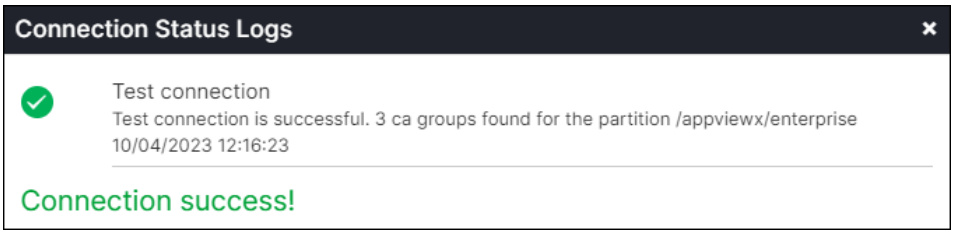


Figure 4. CA Validation



InCommon CA

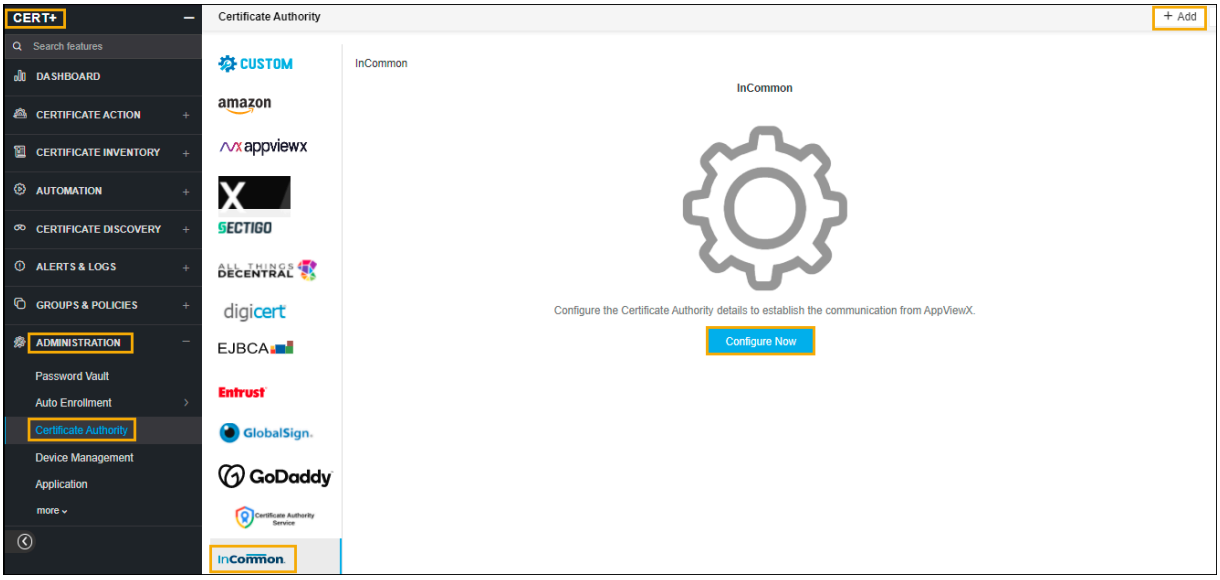
Prerequisites

The prerequisites for configuring InCommon CA account in AppViewX are as follows:

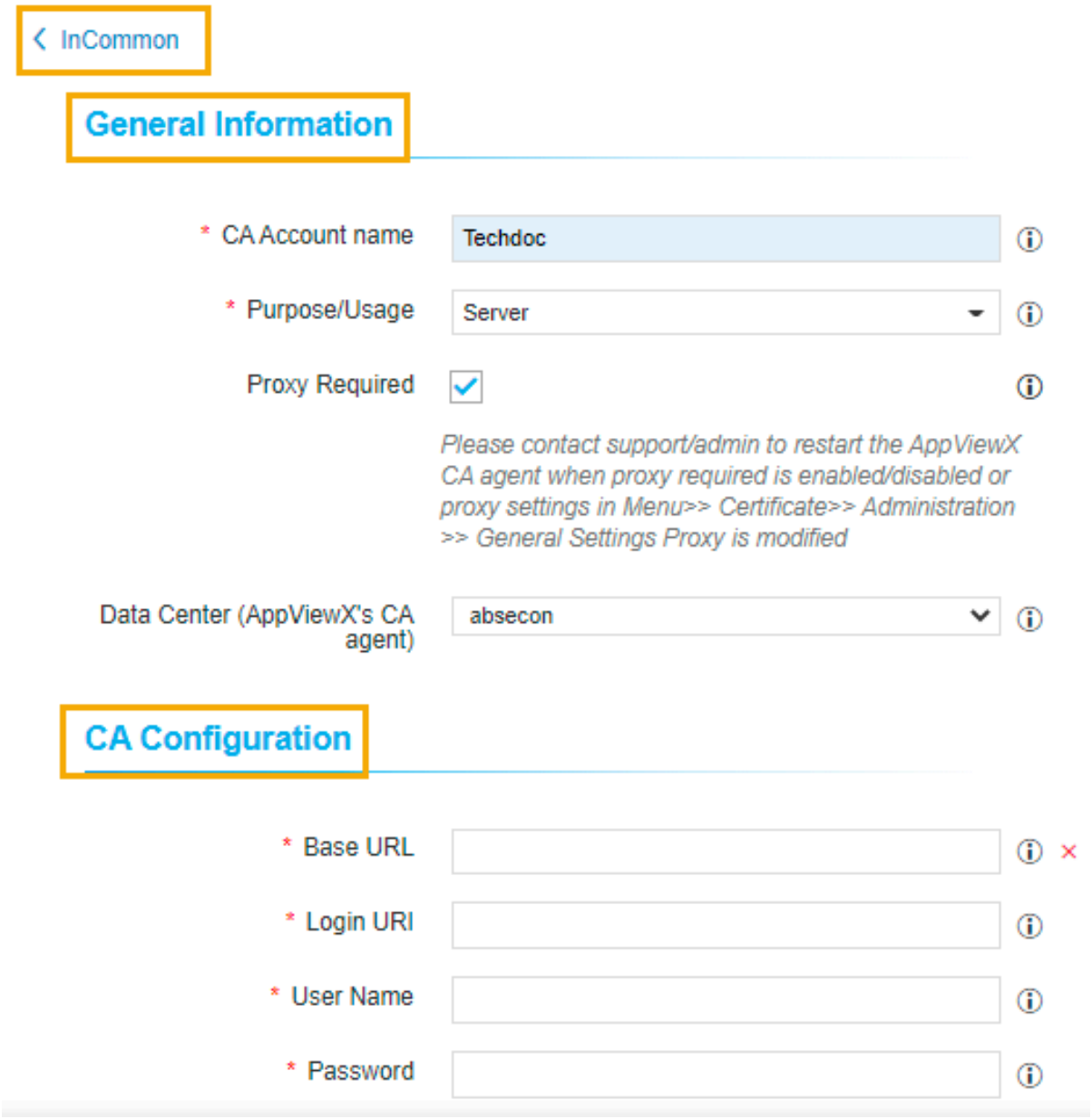
- InCommon Certificate Manager credentials having necessary access for enrolling the certificates.
- AppViewX server should either have internet access or have a proxy configured in AppViewX general settings. Check Proxy Setup for the steps to configure the **proxy. https://adminguide.appviewx.com/proxy-4**
- Username and Password as set up in the Certificate Manager tool.
- An OrgID as provided by InCommon Certificate Manager.
- The login URL and URI.

Configuring InCommon CA

1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Certificate Authority**.
2. From the displayed CA, select **InCommon**.
   The **InCommon** home page is displayed.



3. Click the **Configure Now** button or **+Add** icon from the middle or top-right of the page respectively.
   The **InCommon** configuration page is displayed.



4. Update the following details in the **General Information** section as described in the table:

General Information - Field Description Table

Fields	Description
***CA Account name**	A unique name to identify the CA setting.

Fields	Description
	Note: No special characters other than '.', '-','_' are allowed. The name must not start with special characters.
***Purpose/ Usage**	Certificate Type for which CLM actions will be enabled. Eg. Server, Client.
Proxy Required	Enable this field if the CA communication needs to happen via Proxy. The proxy details configured in general settings will be used for communication.
Data Center (AppViewX's CA agent)	Select the data center through which the CA communication needs to happen.
*: *Mandatory fields*	

5. Update the following details in the **CA Configuration** section as described in the table. These fields are necessary for invoking the InCommon CA APIs for Certificate Management.

CA Configuration - Field Description Table

Fields	Description
***Base URL**	This URL will contain just the hostname of the InCommon CA instance. Eg -https://cert-manager.com/customer/<<customer_uri>>/ssl- here base URL is **https://cert-manager.com.** **Note:** No special characters other than '.', '-','_' are allowed. The name must not start with special characters.
***Login URL**	URI specific to the InCommon CA Customer Account. Eg https://cert-manager.com/customer/<<customer_uri>>/ssl- here URI is **customer_uri**.
***User Name**	User name for the account created with InCommon CA.
***Password**	Password for the account created with InCommon CA.
***Organization ID**	InCommon supports organization hierarchy. Id of the **Organization Unit/ Department** in which Certificates need to be managed has to be specified

Fields	Description
	here. **CLM** actions done using this CA account will be specific to this particular organization's id/department.
*: *Mandatory fields*	

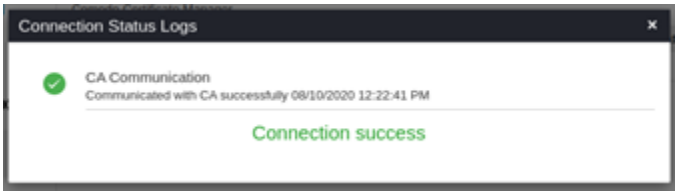
Note: If the certificates from multiple organization's units/departments need to be managed, then a separate CA has to be configured for each organization unit/department in the Incommon CA setting page.

6. Select **Fetch Certificate Types**
   The Certificate types available for the CA account will be fetched from the Certificate Authority.
7. Click **Save**.

Validating InCommon CA

Once the InCommon settings are added validation needs to be done to check whether the connection between AppViewX and InCommon is properly configured.

1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Certificate Authority**.
2. From the displayed CA, select **InCommon**.
   The **InCommon** home page is displayed.
3. In the Status column of the grid with the listed accounts, click **Check** to validate the CA setting that has been created.
   The CA communication will be validated and the **Connection Status** will be shown as either **Success** or **Failure**.



Let’s Encrypt CA

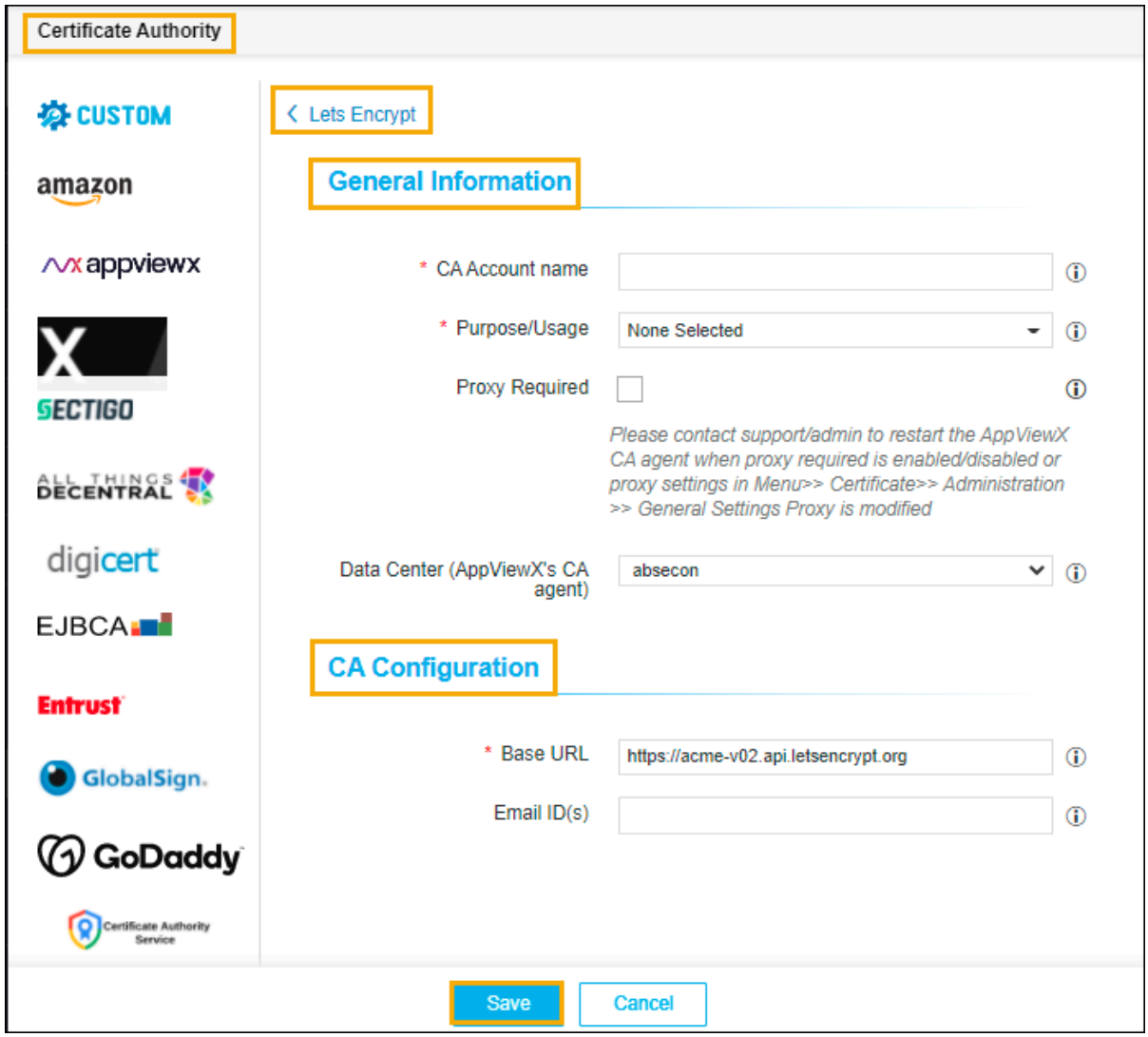
Prerequisites

The prerequisites for configuring Let’s Encrypt CA account in AppViewX are as follows:

- AppViewX server should either have internet access or have a proxy configured in AppViewX general settings. Check Proxy Setup for the steps to configure proxy. https://adminguide.appviewx.com/proxy-4
- Any one of the following Let's Encrypt certificate enrolment URL as per requirement:
  1. https://acme-staging-v02.api.letsencrypt.org for **staging**.
  2. https://acme-v02.api.letsencrypt.org for **production**.

Configuring Let's Encrypt CA

1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Certificate Authority**.
2. From the displayed CA, select **Let's Encrypt**.
   The **Let's Encrypt** home page is displayed.
3. Click the **Configure Now** button or **+Add** icon from the middle or top-right of the page respectively.
   The Let's Encrypt configuration page is displayed.



4. Update the following details in the **General Information** section as described in the table:

General Information - Field Description Table

Fields	Description
***Name**	A unique name to identify the CA setting. **Note:** No special characters other than '.', '-','_' are allowed. The name must not start with special characters.
***Purpose/Usage**	The certificate types will be managed by these settings. For now, Let's Encrypt is having only one purpose **Server**.
Proxy Required	Enable this field if the CA communication needs to happen via Proxy. The proxy details configured in general settings will be used for communication.
Data Center (AppViewX's CA agent)	Select the data center through which the CA communication needs to happen.
*: *Mandatory fields*	

5. Update the following details in the **CA Configuration** section as described in the table. These fields are necessary for invoking the Let's Encrypt CA APIs for Certificate Management.

CA Configuration - Field Description Table

Fields	Description
***Base URL**	Let's Encrypt certificate enrolment URL either staging or production based on the requirement.
***Email ID(s)**	Enter email ID(s) in this field to receive notifications from Let's Encrypt. Multiple email ID must be separated by comma (,).
*: *Mandatory fields*	

6. Click **Save**.

Validating Let’s Encrypt

Once the Let’s Encrypt settings are added validation needs to be done to check whether the connection between AppViewX and Let’s Encrypt is properly configured.

1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Certificate Authority**.
2. From the displayed CA, select **Let's Encrypt**.
   The **Let's Encrypt** home page is displayed.
3. In the Status column of the grid with the listed accounts, click **Check** to validate the CA setting that has been created.
   The CA communication will be validated and the **Connection Status** will be shown as either **Success** or **Failure**.

Microsoft Enterprise CA

Prerequisites

The prerequisites for configuring Microsoft Enterprise CA in AppViewX are as follows:

- AppViewX Windows Gateway installer should be installed in a windows machine, running and reachable from AppViewX vendor plugin through the Communication Modes described below.

Communication Mode Table

Communication mode	Category	Windows gateway machine	Microsoft CA
NATIVE API	User account type	Service account	Service account.
	User permission	NA	Read, Request certificates, Issue and Manage certificates permission at CA level for the service account or the service account group or authenticated users Enroll permission at Certificate template level for the service account or the service account group or authenticated users
	Services	RPC service	RPC service

Communication Mode Table (continued)

Communication mode	Category	Windows gateway machine	Microsoft CA
			certutil.exe command availability
	Ports	NA	135 as the incoming port
POWERSHELL	User account type	Service account	Service account.
	User permission	NA	Full control permission to C:\Windows \Temp Read, Request certificates, Issue and Manage certificates permission at CA level for the service account or the service account group or authenticated users
	Services	RPC Service, WinRM Service, WinRM Configuration, Powershell remoting,certutil.exe command availability	RPC Service, WinRM Service, WinRM Configuration, Powershell remoting,certutil.exe command availability.
	Ports	NA	5985
WMI	User account type	Service account	Service account
	User permission	NA	Full control permission to C:\Windows \Temp Read, Request certificates, Issue and Manage certificates permission at CA level for the service account or the service account group or authenticated users
	Services	WMI service	WMI service

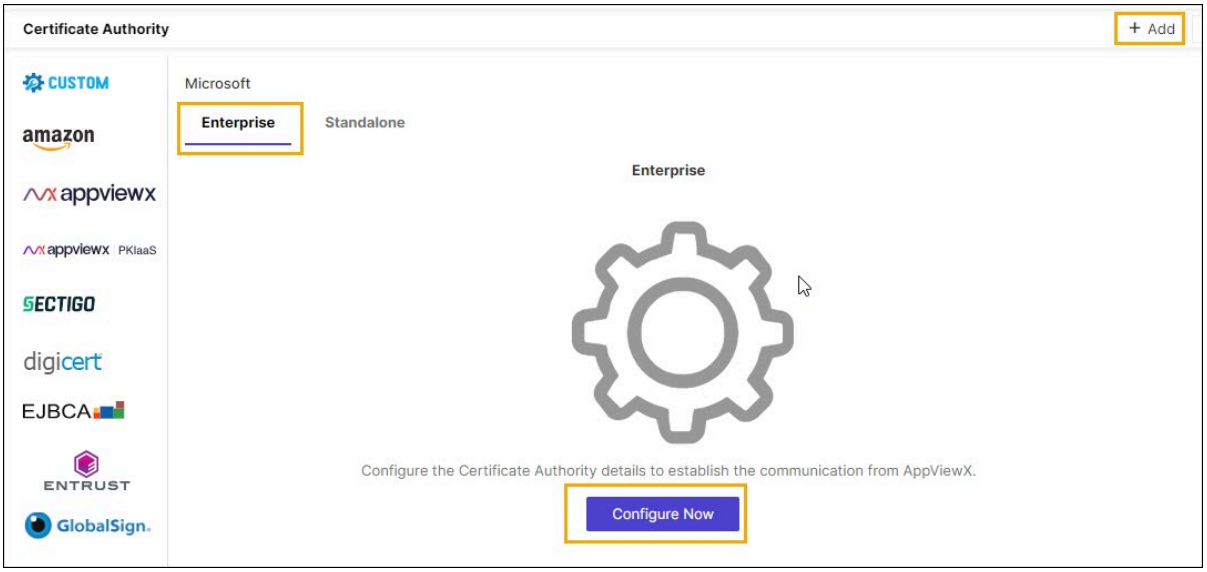
Communication Mode Table (continued)

Communication mode	Category	Windows gateway machine	Microsoft CA
		certutil.exe command availability	certutil.exe command availability
	Ports	NA.	135, 445 or 139

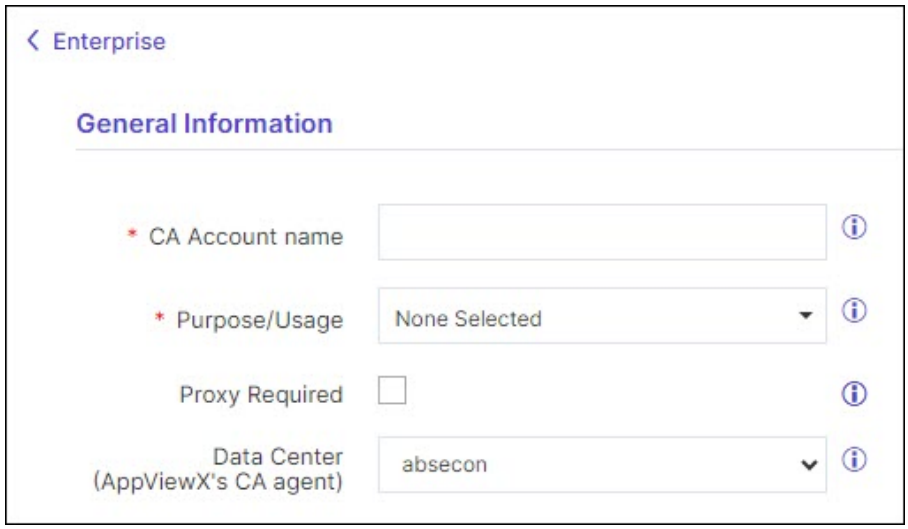
Configuring Microsoft Enterprise CA

1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Certificate Authority**.
2. From the displayed CA, select **Microsoft**.

   The **Microsoft** home page is displayed.



3. Select the **Enterprise** tab.
4. Click the **Configure Now** button or **+Add** icon from the middle or top-right of the page respectively.
5. Update the following details in the **General Information** section as described in the table.



General Information - Field Description Table

Fields	Description
***CA Account name**	A unique name to identify the CA setting. **Note**: No special characters other than '.', '-','_' are allowed. Names should not start with special characters.
***Purpose/Usage**	Certificate Type for which CLM actions will be enabled. Example. Server, Client, Code Signing
Proxy Required	Enable this field if the CA communication needs to happen via Proxy. The proxy details configured in general settings will be used for communication.
Data Center (AppViewX's CA agent)	Select the data center through which the CA communication needs to happen.
*: *Mandatory fields*	

6. Update the following details in the **CA Configuration** section as described in the table.

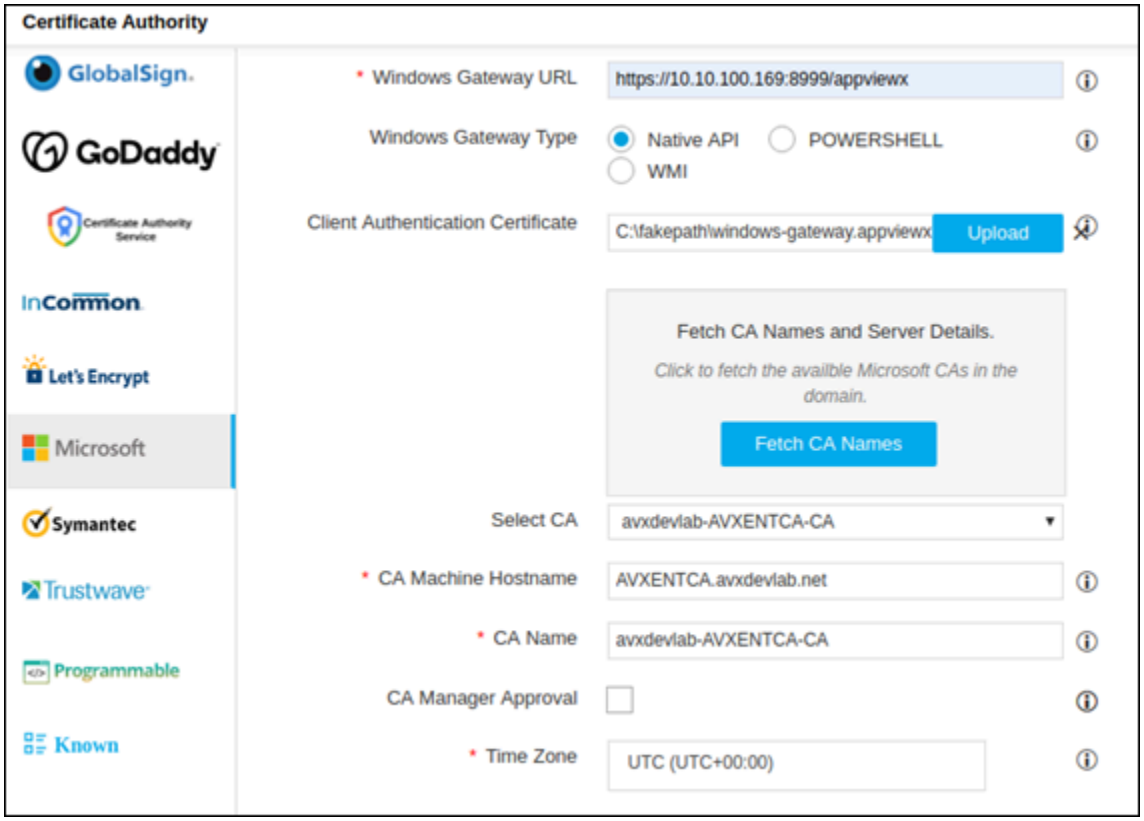
CA Configuration - Field Description Table

Fields	Description
***Windows Gateway URL**	Enter the URL where the AppViewX agent is running.

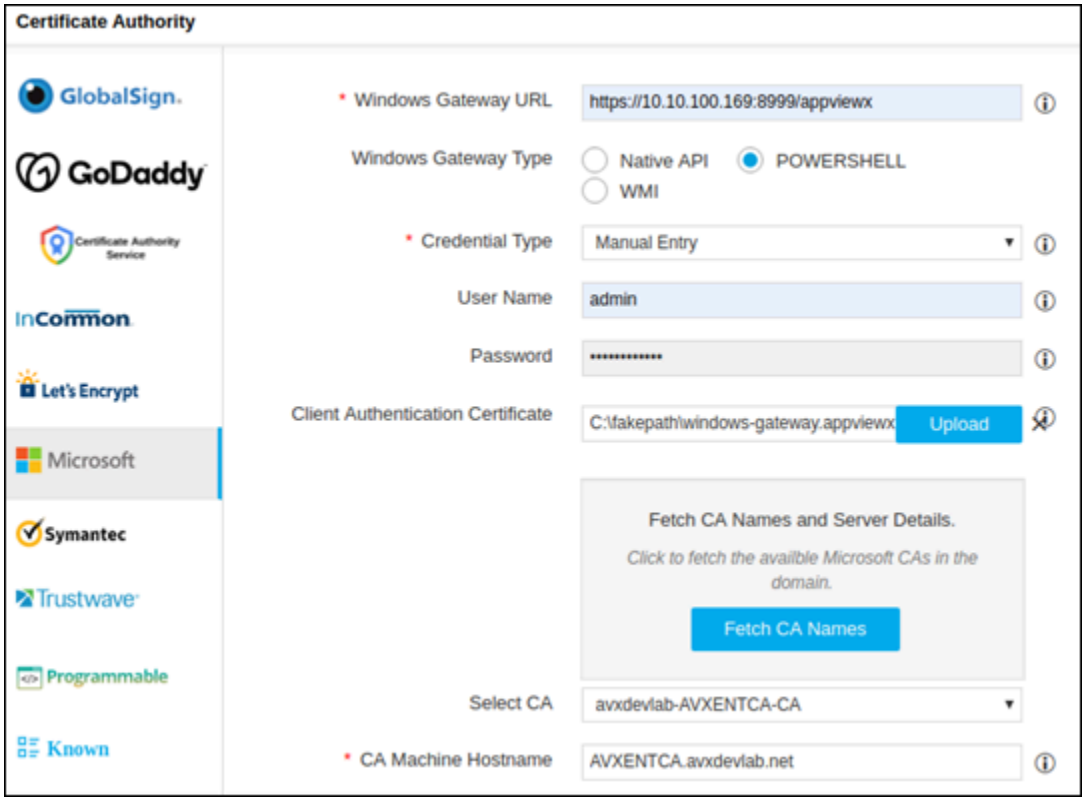
Fields	Description
***Windows Gateway Type**	The mode of communication types from Windows Gateway machine to CA machine. Available types are **NATIVE API, POWERSHELL, WMI**. Refer **Communication Mode**
Client Authentication Certificate	The client certificate used while installing Windows Gateway. Users can use the default client certificate (ClientCertificateGateway.pfx) or the custom certificate given by the Customer.
***Credential Type**	Type of credential to be used. Either **Manual Entry** or **Credential List.**
Username	User name of the credentials.
Password	Password for the username.
*: *Mandatory fields*	

7. Click **Fetch CA Names** to retrieve CAs accessible from Windows Gateway installed machine. Upon successful completion of Fetch CA Names, all reachable CAs listed in **Select CA**.
8. Click on one specific CA and proceed.

Using Native API



Using POWERSHELL / WMI

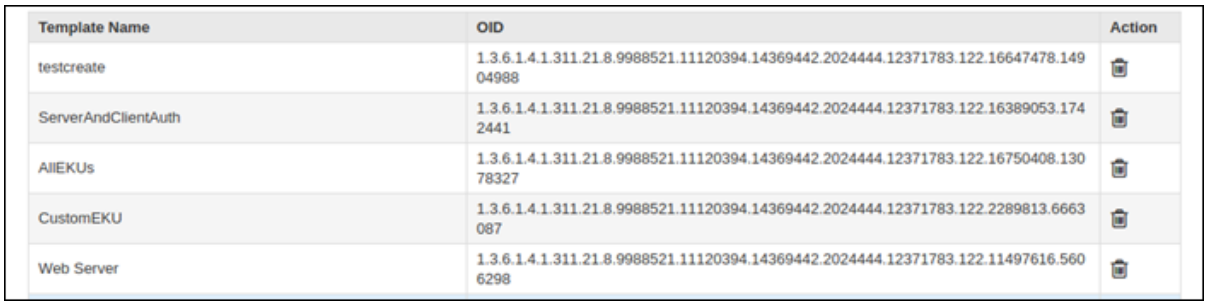


CA Details - Field Description Table

Fields	Description
Select CA	All the reachable CAs are listed here.
***CA Machine Hostname**	Host name of the CA Machine will be auto-filled.
***CA Name**	Name of the CA chosen which will be auto-filled.
CA Manager Approval	Approves the pending enroll / Renew request submitted from AppViewX Certificate.
***Time Zone**	To perform scheduled and Optimized CA discovery, please provide time zone value.
*: *Mandatory fields*	

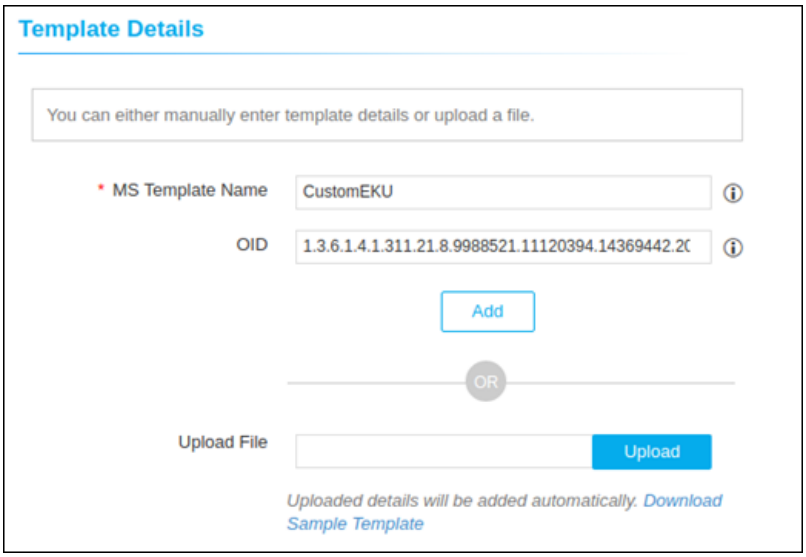
a. Configure the **Template Details**.

Once CA is selected from the **Select CA** list, the **Template** details should have auto-filled as shown below.



Note: If the desired template is not listed, it might not be published in AD. Users can add it manually through MS Template name and OID fields as shown below.

b. In the Template Details section, select/enter the details as shown below.

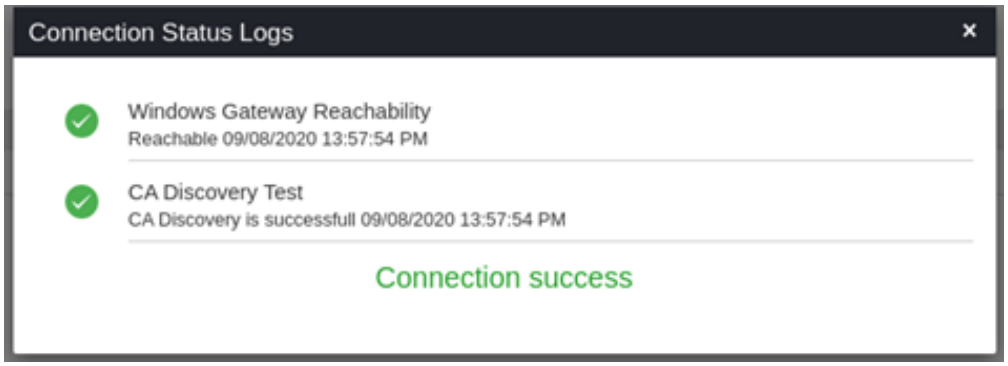


9. Click **Save.**

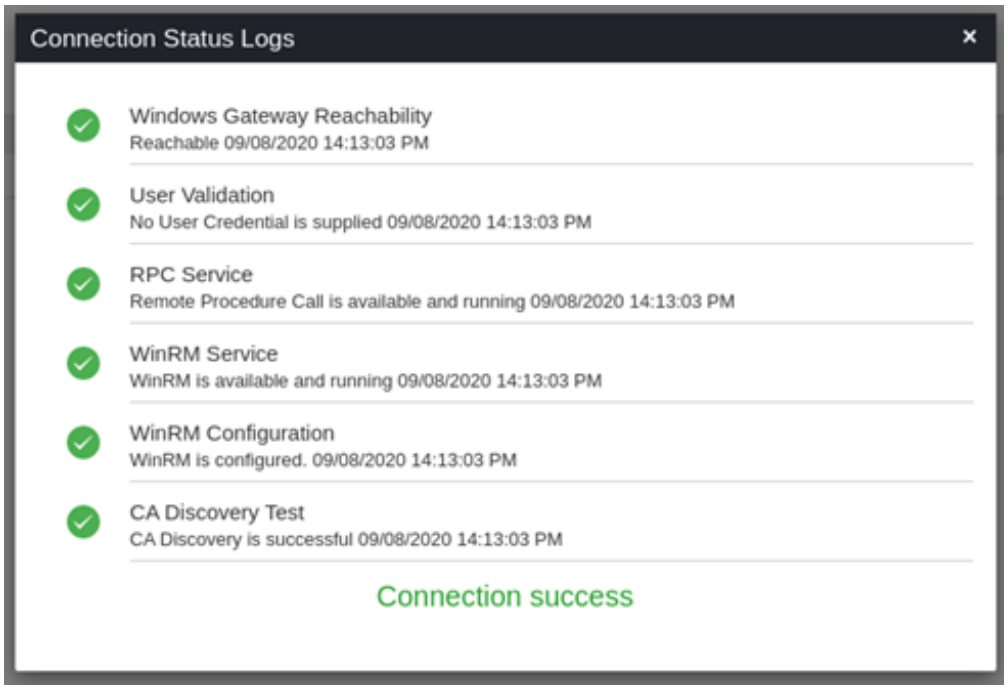
Validating Microsoft Enterprise

Once the Microsoft Enterprise settings are added validation needs to be done to check whether the connection between AppViewX and Microsoft Enterprise is properly configured.

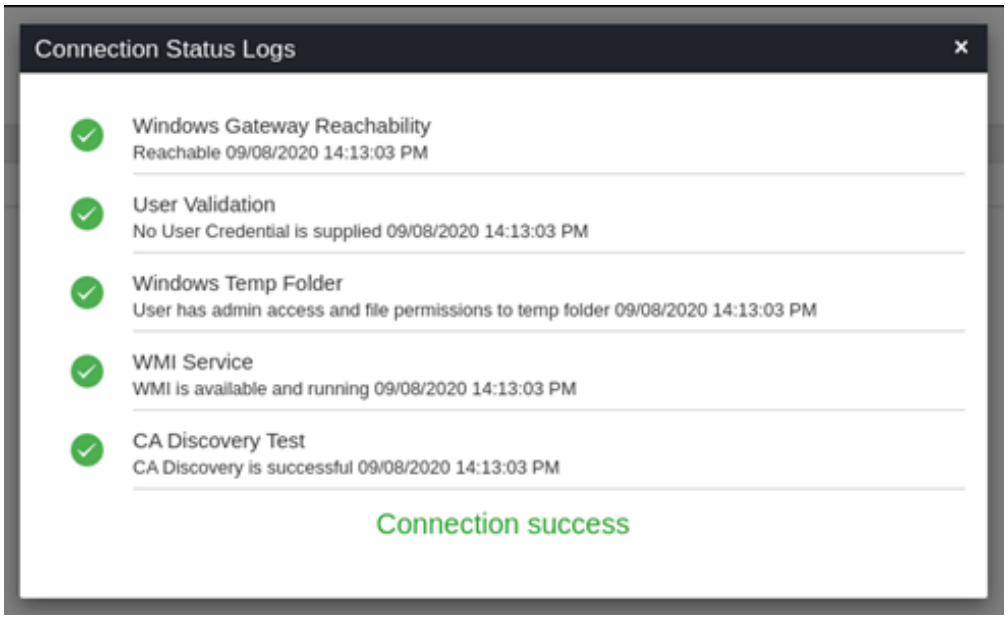
1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Certificate Authority**.
2. From the displayed CA, select **Microsoft**.
   The **Microsoft** home page is displayed.
3. Select the **Enterprise** tab.
4. In the Status column of the grid with the listed accounts, click **Check** to validate the CA setting that has been created.
   The CA communication will be validated and the **Connection Status** will be shown as either **Success** or **Failure**.



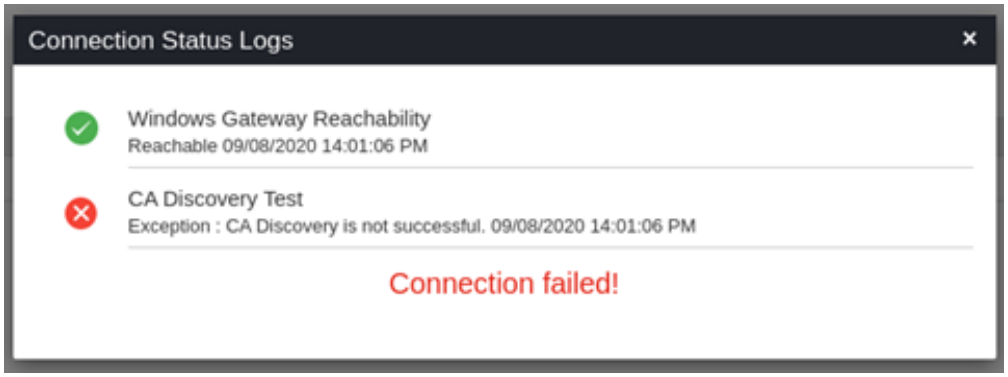
Success Scenario for Native API



Success Scenario for Powershell



Failure scenario for WMI



Microsoft Standalone CA

Prerequisites

The prerequisites for configuring Microsoft Standalone CA in AppViewX are as follows:

- AppViewX Windows Gateway installer should be installed in a windows machine, running and reachable from AppViewX vendor plugin through the Communication Modes described below.

Communication Mode Table

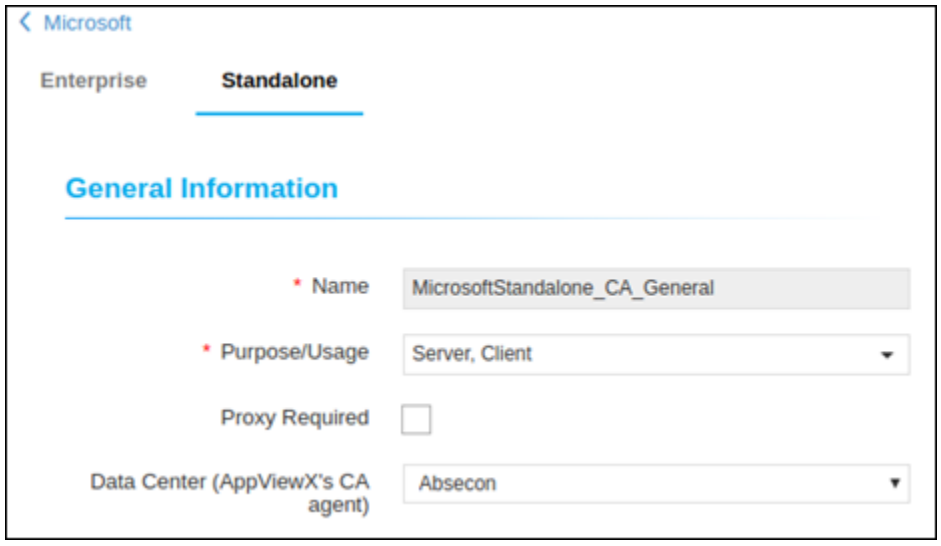
Communication mode	Category	Windows gateway machine	Microsoft CA
NATIVE API	User account type	Service account	Service account
	User permission	NA	Read, Request certificates, Issue and Manage certificates permission at CA level for the service account or the service account group or authenticated users Enroll permission at Certificate template level for the service account or the service account group or authenticated users
	Services	RPC service	RPC service certutil.exe command availability
	Ports	NA	135 as incoming port
POWERSHELL	User account type	Service account	Service account
	User permission	NA	Full control permission to C:\Windows \Temp Read, Request certificates, Issue and Manage certificates permission at CA level for the service account or the service account group or authenticated users
	Services	RPC Service,WinRM Service,WinRM Configuration, Powershell remoting,certutil.exe command availability	RPC Service,WinRM Service,WinRM Configuration, Powershell remoting,certutil.exe command availability
	Ports	NA	5985

Communication Mode Table (continued)

Communication mode	Category	Windows gateway machine	Microsoft CA
WMI	User account type	Service account	Service account
	User permission	NA	Full control permission to C:\Windows \Temp Read, Request certificates, Issue and Manage certificates permission at CA level for the service account or the service account group or authenticated users
	Services	WMI service certutil.exe command availability	WMI service certutil.exe command availability
	Ports	NA	135, 445 or 139

Configuring Microsoft Standalone CA

1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Certificate Authority**.
2. From the displayed CA, select **Microsoft**.
   The **Microsoft** home page is displayed.
3. Select the **Standalone** tab.
4. In the Status column of the grid with the listed accounts, click and then click **+Add** icon or **Configure Now** button.



5. Update the following details in the **General Information** section as described in the table.

General Information - Field Description Table

Fields	Description
Name	A unique name to identify the CA setting. **Note:** No special characters other than '.', '-','_' are allowed. Names should not start with special characters.
***Purpose/Usage**	Certificate Type for which CLM actions will be enabled. For example: Server, Client, and Code Signing.
Proxy Required	Enable this field if the CA communication needs to happen via Proxy. The proxy details configured in general settings will be used for communication.
Data Center (AppViewX's CA agent)	Select the data center through which the CA communication needs to happen.
*: *Mandatory fields*	

6. Update the following details in the **CA Configuration** section as described in the table:

CA Configuration - Field Description Table

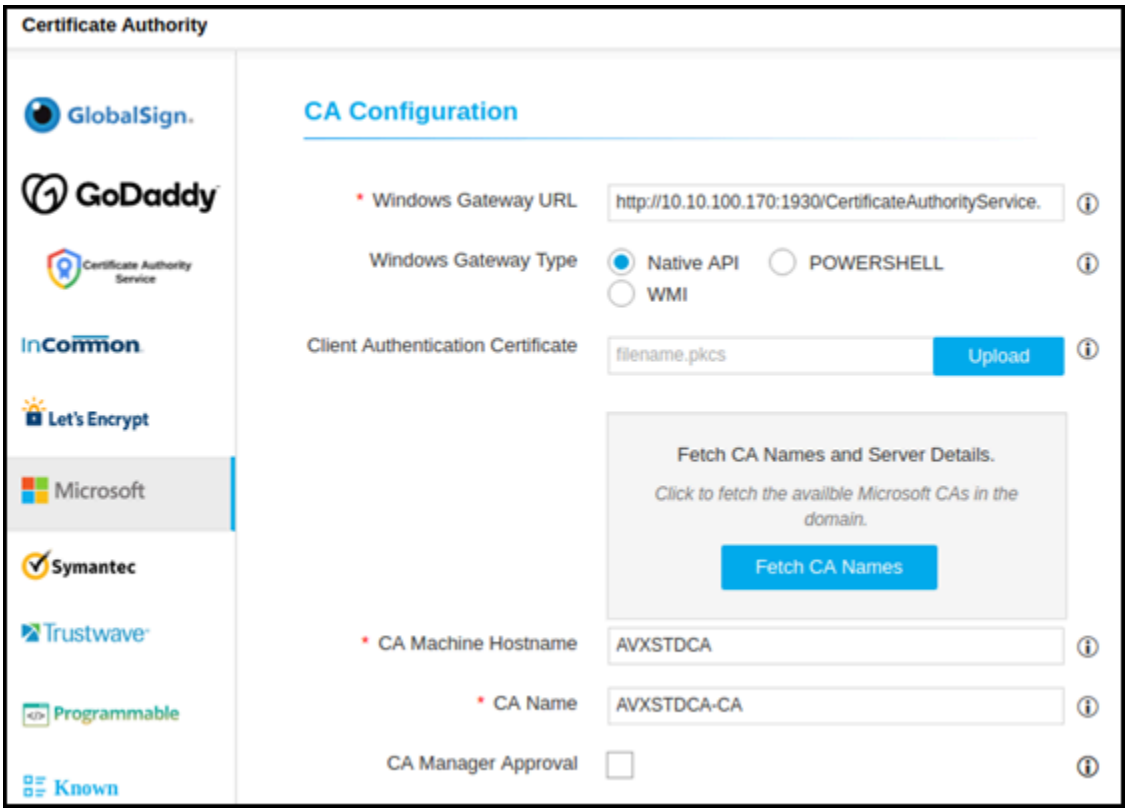
Fields	Description
***Windows Gateway URL**	Enter the URL where the AppViewX agent is running.
***Windows Gateway Type**	The mode of communication types from Windows Gateway machine to CA machine. Available types are **NATIVE API, POWERSHELL, WMI.**
Client Authentication Certificate	The client certificate used while installing Windows Gateway. Users can use the default client certificate (Client Certificate Gateway.pfx) or the custom certificate given by the Customer.
***Credential Type**	Type of credential to be used. Either **Manual Entry** or **Credential List**.
Username	User name of the credentials.
Password	Password for the username.
*: *Mandatory fields*	

7. Click **Fetch CA Names** to retrieve CAs accessible from Windows Gateway installed machine. Upon successful completion of Fetch CA Names, all reachable CAs listed in **Select CA**.
8. Click on one specific CA and proceed.

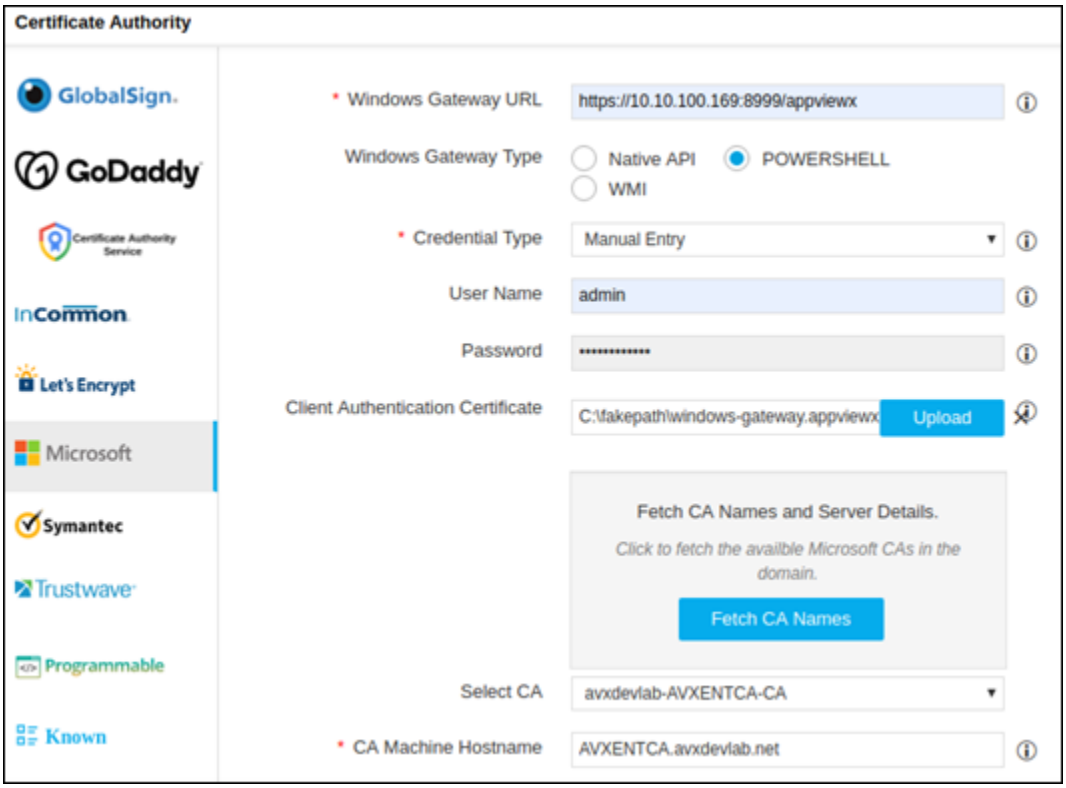
Dynamic Fields for Select CA Section

Fields	Description
Select CA	All the reachable CAs are listed here.
***CA Machine Hostname**	Host name of the CA Machine will be auto-filled.
***CA Name**	Name of the CA chosen which will be auto-filled.
CA Manager Approval	Approves the pending enroll / Renew request submitted from AppViewX Certificate.
*: *Mandatory fields*	

Using Native API



Using Powershell and WMI



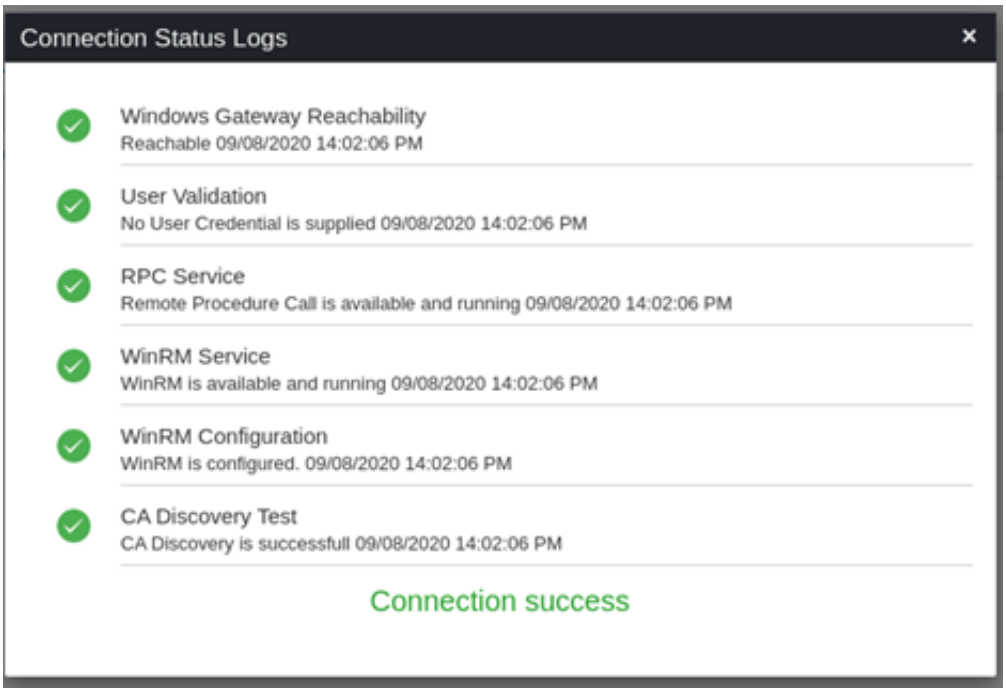
9. Click **Save**.

Validating Microsoft Standalone

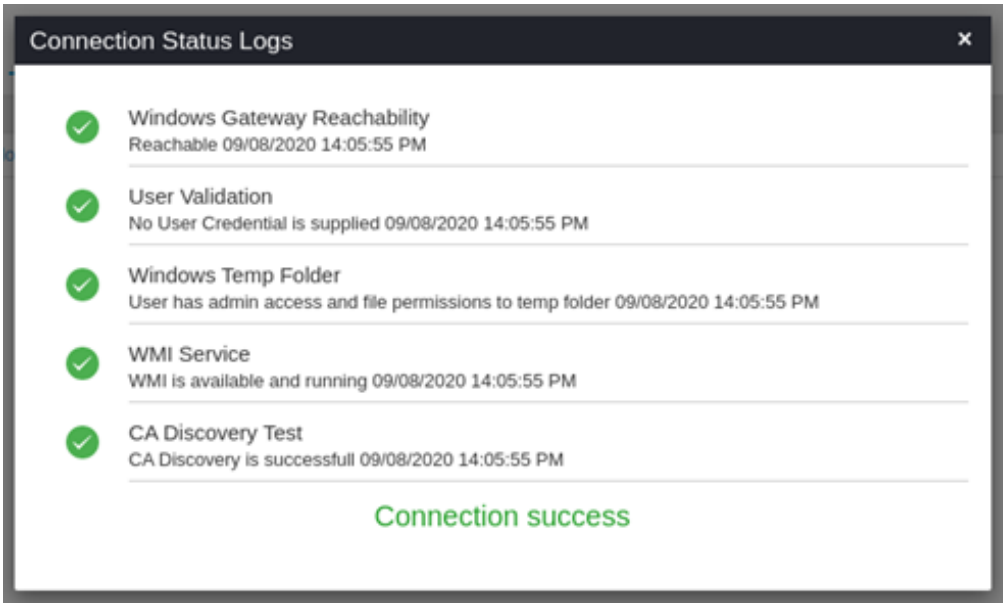
Once the Microsoft Standalone settings are added validation needs to be done to check whether the connection between AppViewX and Microsoft Enterprise is properly configured.

1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Certificate Authority**.
2. From the displayed CA, select **Microsoft**.
   The **Microsoft** home page is displayed.
3. Select the **Standalone** tab.
4. In the Status column of the grid with the listed accounts, click **Check** to validate the CA setting that has been created.
   The CA communication will be validated and the **Connection Status** will be shown as either **Success** or **Failure**.Success scenario for Native APISuccess scenario for PowershellSuccess scenario for WMI.

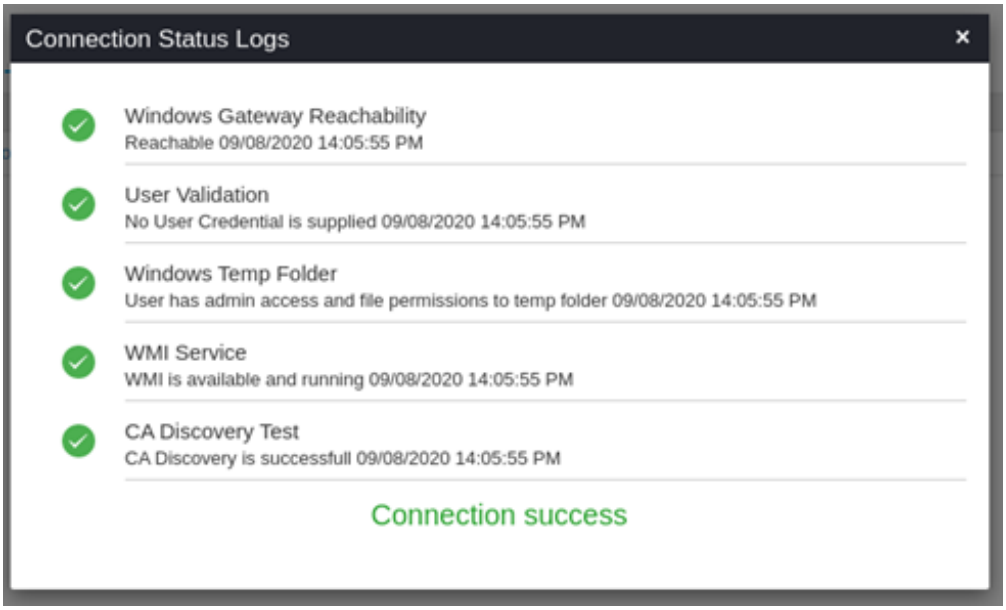
   Success scenario for Native API



Success scenario for Powershell



Success scenario for WMI



Nexus CA

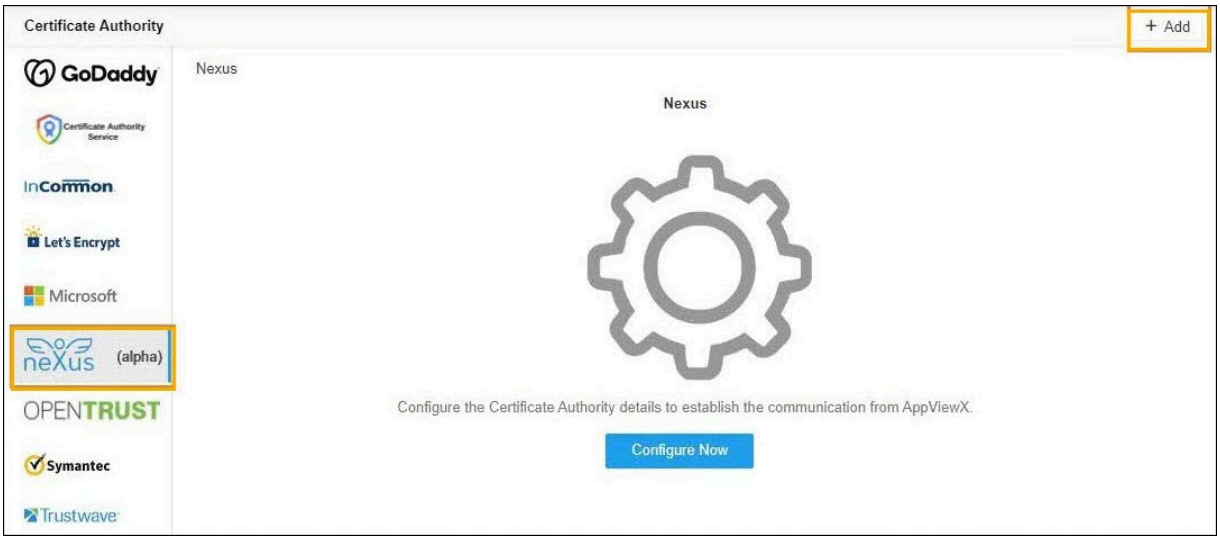
Prerequisites

The prerequisites for configuring a Nexus CA account in AppViewX are as follows:

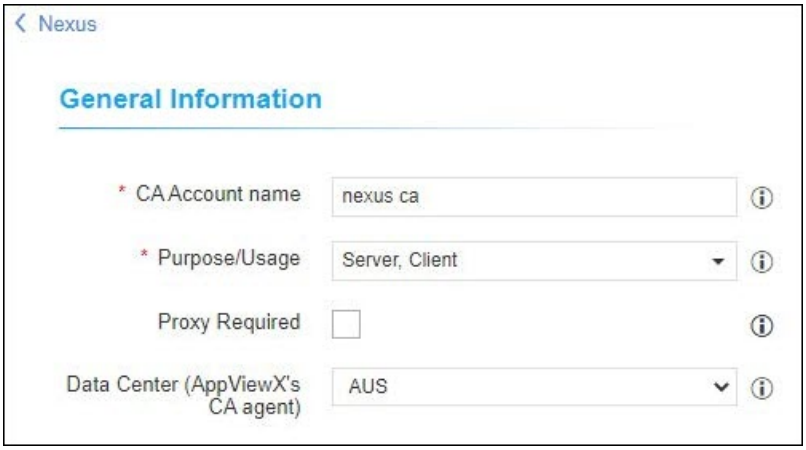
- A Nexus Account with Administrator role Access.
- Before discovery or enrollment, the customer must upload the CA certificates manually.
- The AppViewX server should either have internet access or have a proxy configured in AppViewX general settings.

Configuring Nexus

1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Certificate Authority**.
2. From the displayed CA, select **Nexus**.
   The **Nexus** home page is displayed.



3. Click the **Configure Now** button or **+Add** icon from the middle or top-right of the page respectively. The Nexus configuration page is displayed.
4. Update the following details in the **General Information** section as described in the table.

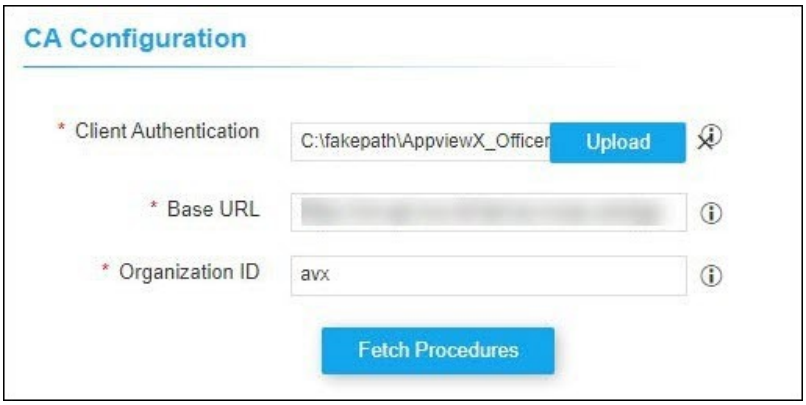


General Information - Field Description Table

Fields	Description
*CA Account name	A unique name to identify the CA setting. No special characters other than '.', '-','_' are allowed. The name should not start with special characters.
*Purpose/Usage	Certificate Type for which CLM actions will be enabled. For example, server and clients
Proxy Required	Enable this field if the CA communication needs to happen via Proxy. The proxy details configured in general settings will be used for communication.

Fields	Description
Data Center (AppViewX's CA agent)	Select the data center through which the CA communication needs to happen.
*: *Mandatory fields*	

5. Update the following details in the **CA Configuration** section as described in the table.



CA Configuration - Field Description Table

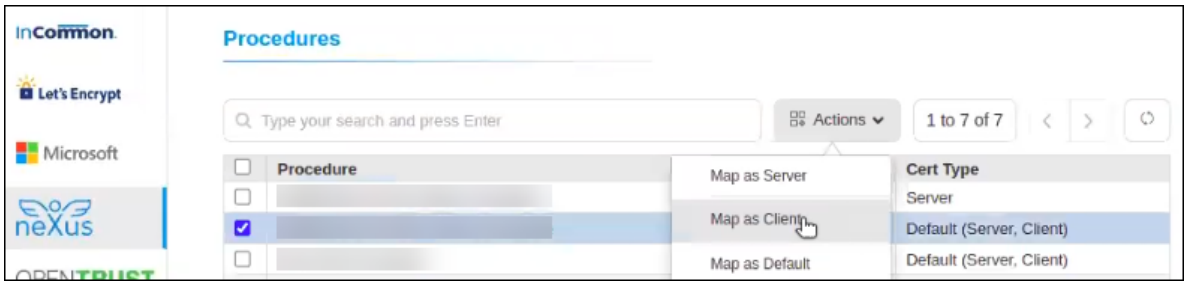
Fields	Description
*SSL URL	Base URL of the SSL API
*User Name	Provide a username of the GCC to communicate with the CA.
*Password	Provide a password for the GCC to communicate with the CA.
*: *Mandatory fields*	

6. Select **Fetch Procedures**.

   The procedures available in the Nexus CA account will be fetched and listed for the specific user.

7. To map the fetched procedures, click on one or many and click the **Actions** dropdown

- **CASE 1** - If the user selects Server only in Purpose and Usage, then the fetched procedure by default will be of server/client both. The Action dropdown will only have - **Map as Server**. and **MAP as Default**
- **CASE 2** - If the user selects Client only in Purpose and Usage, then the fetched procedure by default will be of server/client both. The Action dropdown will only have - **Map as Client**. and **MAP as Default**
- **CASE 3** - If the user selects Server and Client both in Purpose and Usage, then the fetched procedure by default will be of server/client both. The Action dropdown will have both the actions **Map as Client** , **Map as Server**, and **MAP as Default**

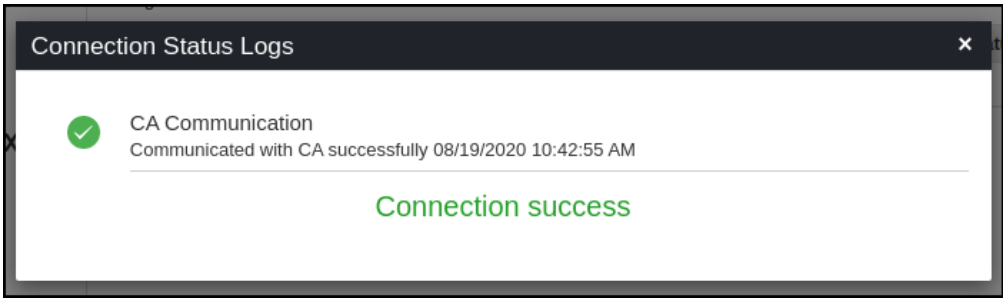


8. Click **Save**.

Validating Nexus

Once the Nexus settings are added, the validation must be done to check whether the connection between AppViewX and Nexus is configured properly.

1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Certificate Authority**.
2. From the displayed CA, select **Nexus**.
3. In the Status column of the grid with the listed accounts, click **Check** to validate the CA setting that is created.
   The CA communication will be validated and the Connection Status will be shown as either Success or Failure.



Sectigo CA

Prerequisites

The prerequisites for configuring a Sectigo CA account in AppViewX are as follows:

1. To create a CA configuration the following values are required:
   - Base URL
   - Login URI
   - Username (The new administrator's login name. Refer point 2.)
   - Password
   - Organization ID (Refer point 4.)

   Once your organization has subscribed for a Sectigo account, you will be provided with a **Username**, **Password**, and **Login URL** for SCM (Sectigo Certificate Manager). The default format of this URL is https://cert-manager.com/customer/<customer URI>/, where **<customer URI>** is a path segment specific to your company.
2. The Username and Password should be of the following administrators:
   - Master Registration Authority Officer (MRAO)
   - Registration Authority Officer (RAO)
3. The above administrators should have the following privileges.

Privileges	
Allow SSL details changing	Enables the new MRAO, RAO SSL, and DRAO SSL to change the details of SSL certificates by navigating to Certificates > SSL Certificates.
Allow SSL auto approve	SSL certificates requested by the MRAO are automatically approved, and those requested by a RAO SSL and DRAO SSL are automatically approved by the administrator of same level and await approval from higher level administrator.

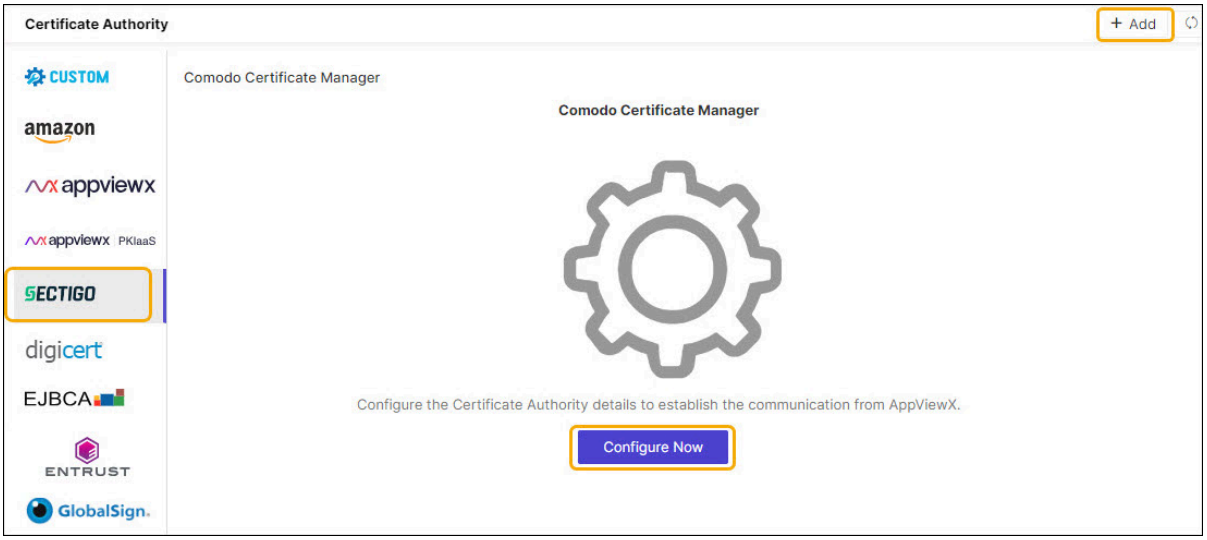
   To review the administrator details in the SCM, navigate to **Settings > Admins**, select the administrator in the list, and click **Edit**. This displays the Edit Client Admin dialog, Add/Edit the necessary privileges and click **Save**.
4. **Organization Id:** Organizations are umbrella entities created by administrators for the purposes of requesting, issuing, and managing certificates for domains and employees. The Organizations page is used to add and modify the organizations.

   To review the organization details in the SCM, navigate to **Organizations**, select the organization in the list, and click **Edit**. This displays the Edit Organization dialog shown in the following illustration.

Configuring Sectigo CA

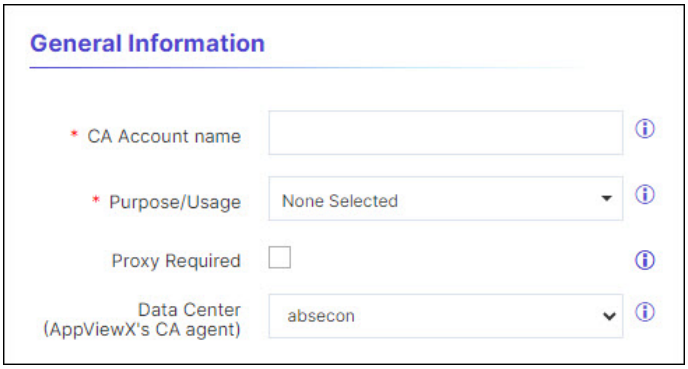
1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Certificate Authority**.
2. From the displayed CA, select **Symantec**.

   The **Sectigo** home page is displayed.



3. (Optional if creating for the first time) Select the **Comodo Certificate Manager** tab.
4. Click the **Configure Now** button or **+Add** icon from the middle or top-right of the page respectively.

   The Sectigo CA details page is displayed.
5. Update the following details in the **General Information** section as described in the table:

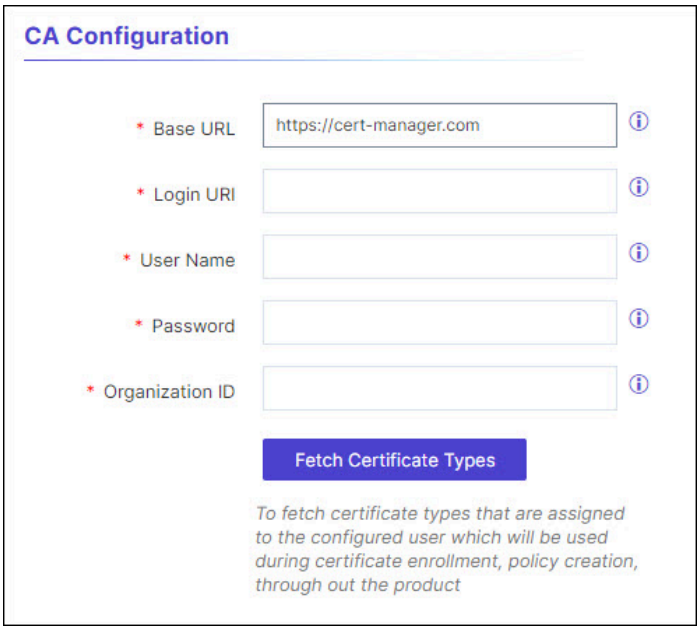


General Information - Field Description Table

Fields	Description
***CA Account name**	A unique name to identify the CA setting. **Note**: No special characters other than '.', '-','_' are allowed. Names should not start with special characters.

Fields	Description
***Purpose/Usage**	Certificate Type for which CLM actions will be enabled. *Example*: Server and Client.
Proxy Required	Enable this field if the CA communication needs to happen via Proxy. The proxy details configured in general settings will be used for communication.
Data Center (AppViewX's CA agent)	Select the data center through which the CA communication needs to happen.
*: *Mandatory fields*	

6. Update the following details in the **CA Configuration** section as described in the table. These fields are necessary for invoking the Sectigo CA APIs for Certificate Management.



CA Configuration - Field Description Table

Fields	Description
***Base URL**	This URL will contain the hostname of the Sectigo CA instance and used for constructing the API requests.
***Login URI**	Provide the customer login URI for API authentication.
***User Name**	Enter the Username of the Sectigo portal to communicate with the CA.

Fields	Description
***Password**	Enter the Password of the Sectigo portal to communicate with the CA.
***Organization Id**	Enter the organization id used for the certificate lifecycle action. (You will find it in the Organization tab of the Sectigo portal)
*: *Mandatory fields*	

7. Click **Fetch Certificate Types**

   The certificate types that are assigned to the configured user which will be used during certificate enrollment, policy creation, through out the product.
8. Update the following details in the **Advanced Settings** section as described in the table.

Advanced Settings - Field Description Table

Fields	Description
Poll after CSR Submission	A check box field when selected will fetch the certificated immediately after CSR Submission on enrollment, renew, and reissue of certificate with the retry count and retry frequency as described below.
***Retry Count**	The number of times the polling will take place after CSR submission. Enter a value between 1 and 10.
***Retry Frequency**	The duration of the polling. enter the value between 1 and 30seconds
*: *Mandatory fields*	

9. Click **Fetch Custom Attributes**.

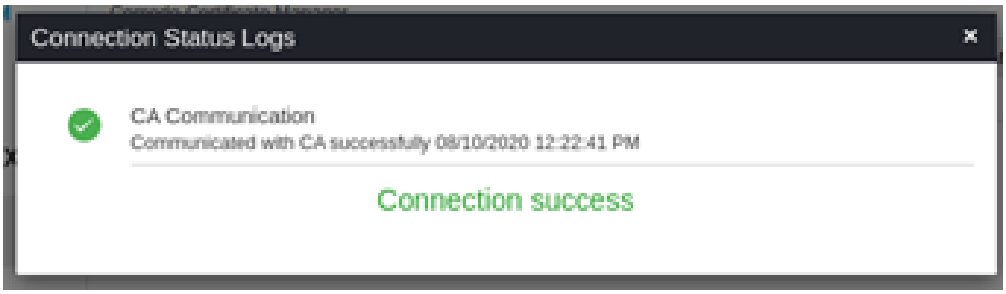
   The attributes available for the CA account will be fetched from the Certificate Authority along with the CA and profile names. A pop-up message is displayed as **CA and profiles fetched**.
10. Click **Save**.

    The created Sectigo configuration settings will be added. The pop-up message is displayed as **<CA_name> Settings Added**.

Validating Sectigo CA

Once the Sectigo settings are added, validation needs to be done to check whether the connection between AppViewX and Sectigo is properly configured.

1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Certificate Authority**.
2. From the displayed CA, select **Sectigo**.
3. In the Status column of the grid with the listed accounts, click **Check** to validate the CA setting that has been created.
   The CA communication will be validated and the **Connection Status** will be shown as either **Success** or **Failure**.



SwissSign CA

Prerequisites

The following values are needed to configure the SwissSign CA:

1. CA Base URL (the value is available in the UI field of the CA setting page)
2. Username
3. API key
4. AppViewX server should either have internet access or have a proxy configured in AppViewX general settings. Refer to the section Managing Proxy Settings in the Platform guides.

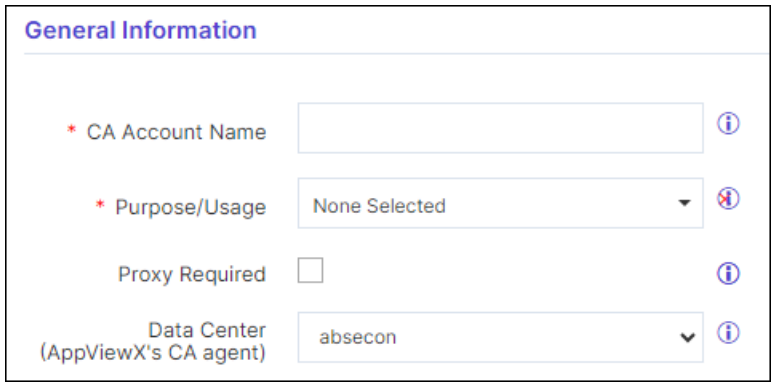
SwissSign issues the Username and API key. The SwissSign team manages user creation and access controls for users. Users must have a level of **RAO (Registration Authority Officer)** or higher to perform actions like issuing, revoking, updating, and viewing certificates.

Configuring SwissSign CA

1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Certificate Authority**.
2. From the displayed CA, Select **SwissSign**.

The **SwissSign** home page is displayed.

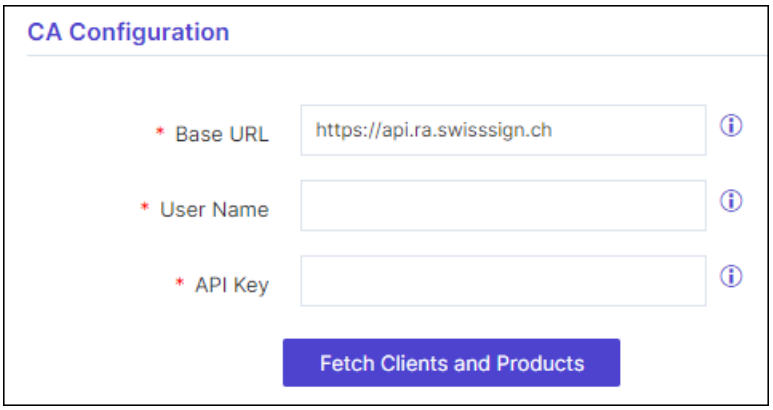
3. Click the **Configure Now** button or **+Add** icon from the middle or top-right of the page respectively.
   The **SwissSign CA** configuration page is displayed.
4. Update the following details in the **General Information** section as described in the table:



General Information - Field Description Table

Fields	Description
***CA Account name**	Enter a unique name in the text field to identify the CA account and be represented during certificate enrollment and policy creation. No special characters other than '.', '-','_' are allowed. Names should not start with special characters.
***Purpose/Usage**	Select the purpose of the certificate from the dropdown list for which CLM actions will be enabled. Example: Server, Client
Proxy Required	Enable this field if the CA communication needs to happen via Proxy. The proxy details configured in general settings will be used for communication.
Data Center (AppViewX's CA agent)	Select the data center through which the CA communication needs to happen.
*: *Mandatory fields*	

5. Update the following details in the **CA Configuration** section as described in the table. These fields are necessary for invoking the CSC Global CA APIs for Certificate Management:

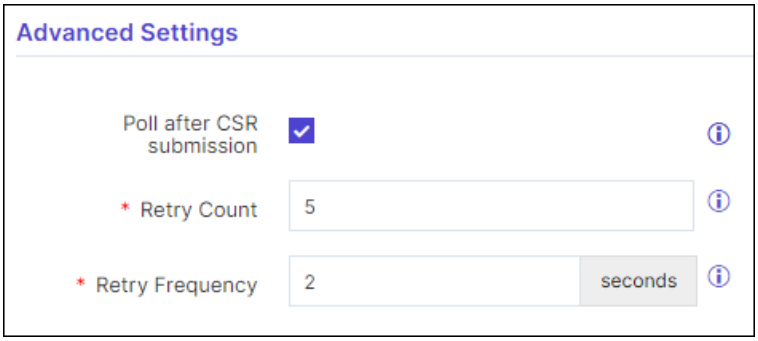


CA Configuration - Field Description Table

Fields	Description
***CA Base URL**	Enter the base URL that will be used to construct the API request. The value is *https://api.ra.swisssign.ch*
***Username**	Enter the service account username to communicate with the RA.
***API Key**	Enter the API key which is a unique identifier used to authenticate the user. The value entered will be masked and encrypted.
*: *Mandatory fields*	

6. Click **Fetch Clients and Products.**

   If the data provided is accurate the success message “Clients and Products fetched successfully“ will be displayed and a table is displayed with two columns - *Client* and *Product Name* (the product name is similar to the certificate profile).
7. Update the following details in the **Advanced Settings** section as described in the table.



Advanced Settings - Field Description Table

Fields	Description
***Poll after CSR submission**	Selecting the checkbox enables polling after CSR submission. It fetches the certificates immediately after CSR submission on enrollment, renew, and reissue with the retry count and retry frequency specified in the fields below.
***Retry Count**	Enter a value between 1 - 10.
***Retry Frequency**	Enter a value between 1 - 30 seconds.
*: *Mandatory fields*	

8. Click **Save**.
   In case of success the message “CA setting updated successfully“ is displayed and for failure “CA setting update failed“ is displayed.

> **Note:** In case “Fetch Clients and Products” are not triggered before saving CA settings, this action can explicitly perform both these actions and update the CA.

Validating SwissSign CA

Once the SwissSign CA settings are added, validation needs to be done to check whether the connection between AppViewX and SwissSign is properly configured.

1. On the SwissSign CA page, the above configured CA will be displayed with a **Check** button in the last column of the table.
2. Click **Check** to validate the CA setting that is created.
   The CA communication will be validated and the **Connection Status** will be shown as either **Success** or **Failure**.

Symantec CA

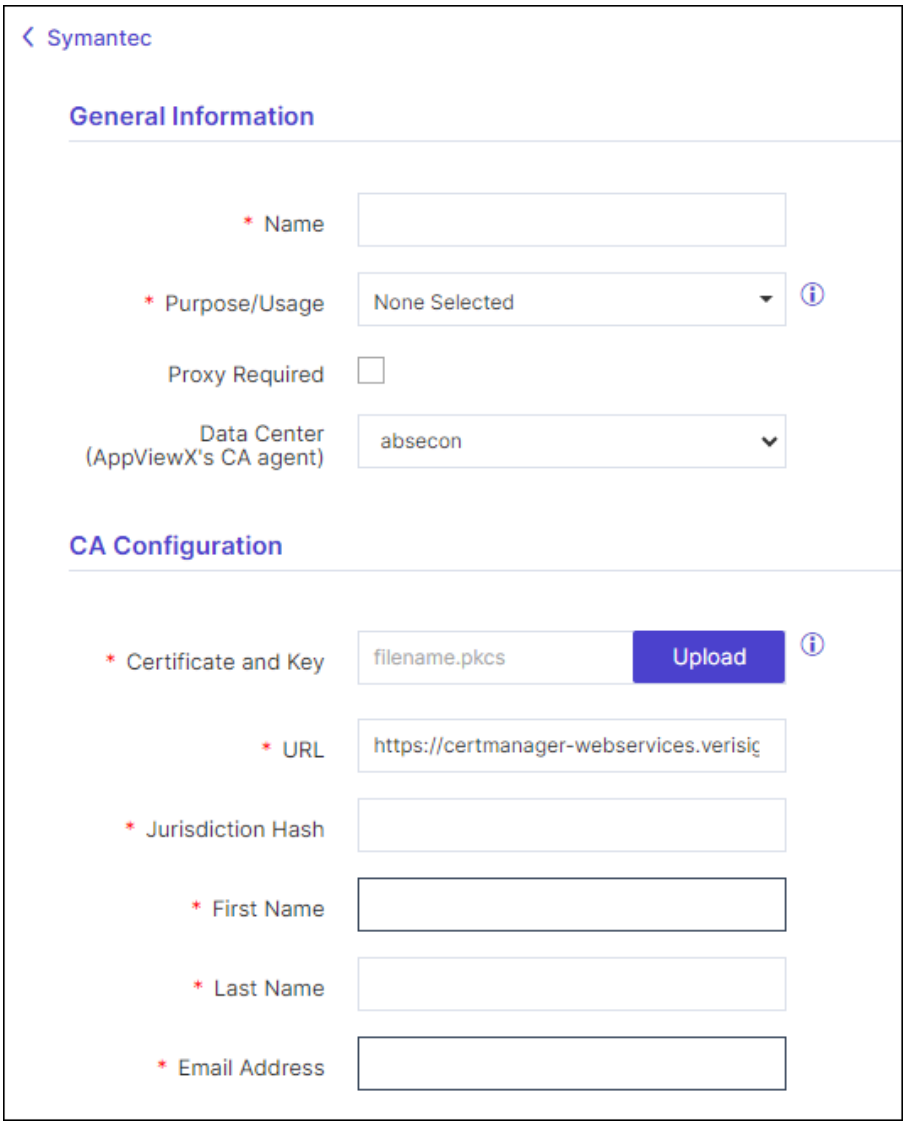
Prerequisites

The prerequisites for configuring a Symantec account in AppViewX are as follows:

- A Symantec client certificate for a user having the necessary access for enrolling the certificates and other Certificate Lifecycle Management(CLM) operations.
- AppViewX server should either have internet access or have a proxy configured in AppViewX general settings. Check Proxy Setup for the steps to configure proxy. https://adminguide.appviewx.com/proxy-4
- Symantec users should be associated with the role **“w=VICE2 web services application”**.
- Required organization status should be “valid”.
- If the EV certificate type is enabled, then the EV status of the organization should be “Yes”.
- The required domain should be registered with the organization.
- The required certificate types should be enabled with the required values in the portal.
- Unit values should be available for the required certificate type.

Configuring Symantec CA

1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Certificate Authority**.
2. From the displayed CA, select **Symantec**.
   The **Symantec** home page is displayed.
3. Click the **Configure Now** button or **+Add** icon from the middle or top-right of the page respectively.
   The **Symantec** configuration page is displayed.



4. Update the following details in the **General Information** section as described in the table.

General Information - Field Description Table

Fields	Description
***CA Account name**	A unique name to identify the CA setting. **Note:** No special characters other than '.', '-','_' are allowed. The name must not start with special characters.
***Purpose/Usage**	Certificate Type for which CLM actions will be enabled. For example, Server and Client.

Fields	Description
Proxy Required	Enable this field if the CA communication needs to happen via Proxy. The proxy details configured in general settings will be used for communication.
Data Center (AppViewX's CA agent)	Select the data center through which the CA communication needs to happen.
*: *Mandatory fields*	

5. Update the following details in the **CA Configuration** section as described in the table. These fields are necessary for invoking the Symantec CA APIs for Certificate Management.

CA Configuration - Field Description Table

Fields	Description
***Certificate and Key**	Client authentication certificate for API communication. **Note:** Must be a valid <.p12> or <.pfx> file.
***URL**	Symantec URL used for API communications. For example, **https://certmanager-webservices.websecurity.symantec.com/vswebservices/**
***Jurisdiction hash**	Jurisdiction hash of the Symantec account. Available in the top right corner of the Symantec portal.
***First name**	First name of the user.
***Last name**	Last name of the user.
*: *Mandatory fields*	

6. Click **Save**.

Validating Symantec

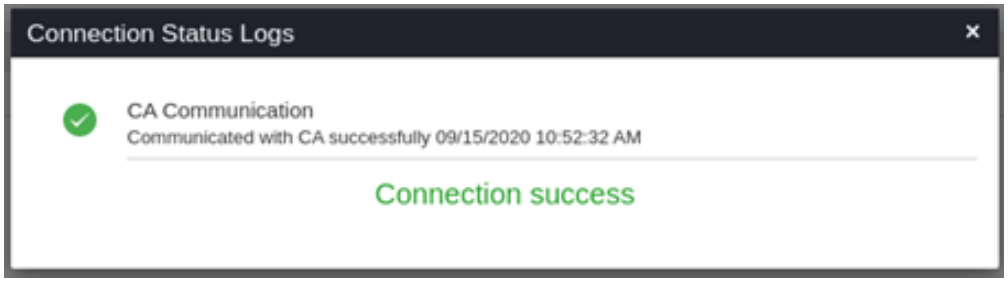
Once the Symantec settings are added validation needs to be done to check whether the connection between AppViewX and Symantec is properly configured.

1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Certificate Authority**.
2. From the displayed CA, select **Symantec**.

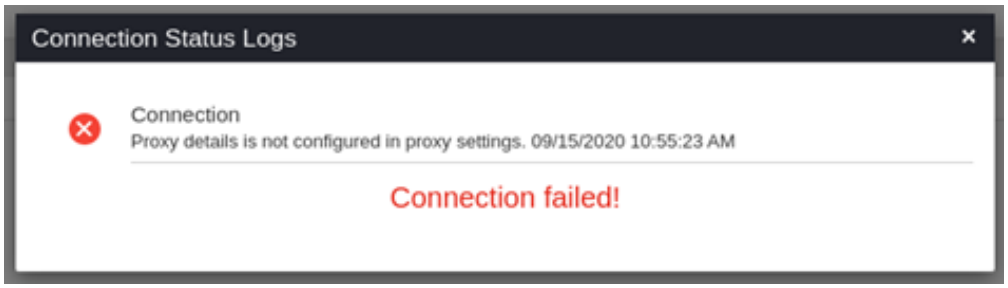
The **Symantec** home page is displayed.

3. In the Status column of the grid with the listed accounts, click **Check** to validate the CA setting that has been created.
   The CA communication will be validated and the **Connection Status** will be shown as either **Success** or **Failure**.

Success Scenario



Failed Scenario



Trustwave CA

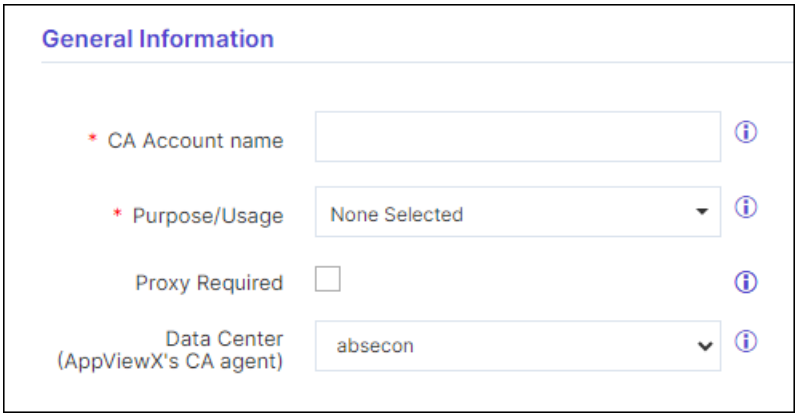
Prerequisites

The prerequisites for configuring Trustwave CA account in AppViewX are as follows:

- Trustwave API URL. Ex: https://testapi.ssl.trustwave.com/3.0/
- Reachability from AppViewX southbound to Trustwave API URL via proxy or direct internet connection
- Valid credentials for communicating to Trustwave CA via API
- Reseller id
- Account details provided inTrustwave account such as Organization Name, Email address, Organization Address, City, State, Zip code, Country, Phone number
- AppViewX server should either have internet access or have a proxy configured in AppViewX general settings.

Configuring Trustwave CA

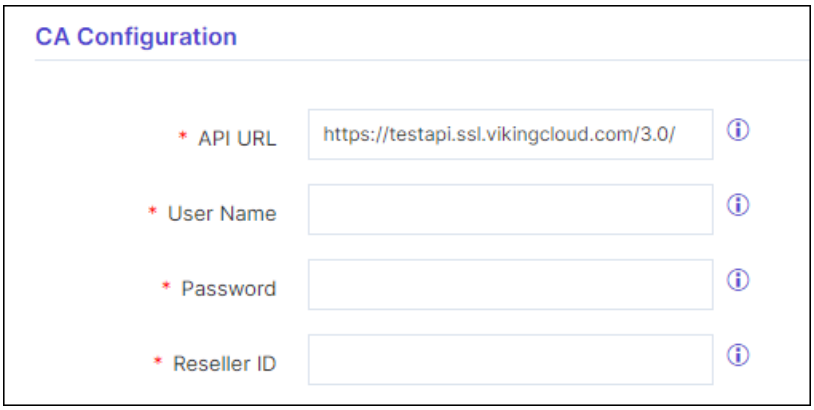
1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Certificate Authority**.
2. From the displayed CA, select **Trustwave**.
   The **Trustwave** home page is displayed.
3. Click the **Configure Now** button or **+Add** icon from the middle or top-right of the page respectively.
   The Trustwave CA details page is displayed.
4. Configure the **General Information** details as follows:



General Information - Field Description Table

Fields	Description
***CA Account name**	Provide an account name for the CA setting.
***Purpose/Usage**	Choose the certificate categories that will be managed by this setting. Possible certificate categories could be: a. Server b. Code Signing
Proxy Required	Enable this field if the CA communication needs to happen via **Proxy**.
Data Center (AppViewX's CA agent	Choose the appropriate **Data Center**.
*: *Mandatory fields*	

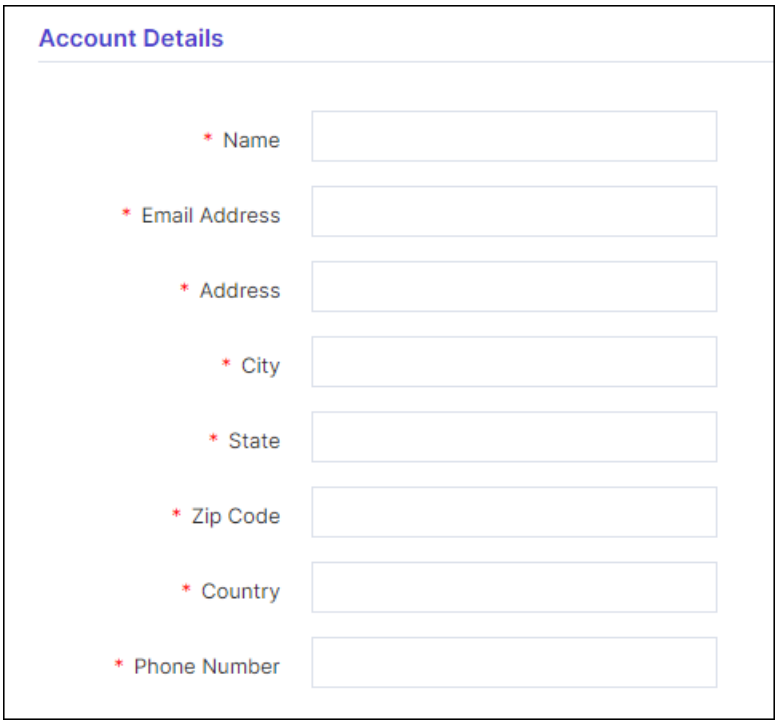
5. Configure the **CA Configuration** with information you want to configure:



CA Configuration - Field Description Table

Fields	Description
***API URL**	The Trustwave API URL to communicate. E.g.: https:// testapi.ssl.trustwave.com/3.0/
***Username**	The username for API authentication.
***Password**	The password for API authentication.
***Reseller ID**	The Reseller Id for the account.
**: Mandatory fields*	

6. Configure the **Account Details** with information you want to configure:



CA Configuration - Field Description Table

Fields	Description
***Name**	The Organization name given in the Trustwave account.
***Email Address**	The Administrator or organization email address given in the Trustwave account.
***Address**	The Organization Address given in the Trustwave account.
***City**	The city name given in the Trustwave account.
***State**	The state name given in the Trustwave account.
***Zip code**	The zip code given in the Trustwave account.
***Country**	The country code given in the Trustwave account. E.g.: US.
***Phone number**	The phone number given in the Trustwave account.
*: *Mandatory fields*	

7. Click **Save**.

Validating Trustwave

Once the Trustwave settings are added validation needs to be done to check whether the connection between AppViewX and Trustwave is properly configured.

1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Certificate Authority**.
2. From the displayed CA, select **Trustwave**.
3. In the Status column of the grid with the listed accounts, click **Check** to validate the CA setting that is created.
   The CA communication will be validated and the **Connection Status** will be shown as either **Success** or **Failure**.

Certificate Group

Before you Begin

Know the following before starting the **Certificate Groups** configuration:

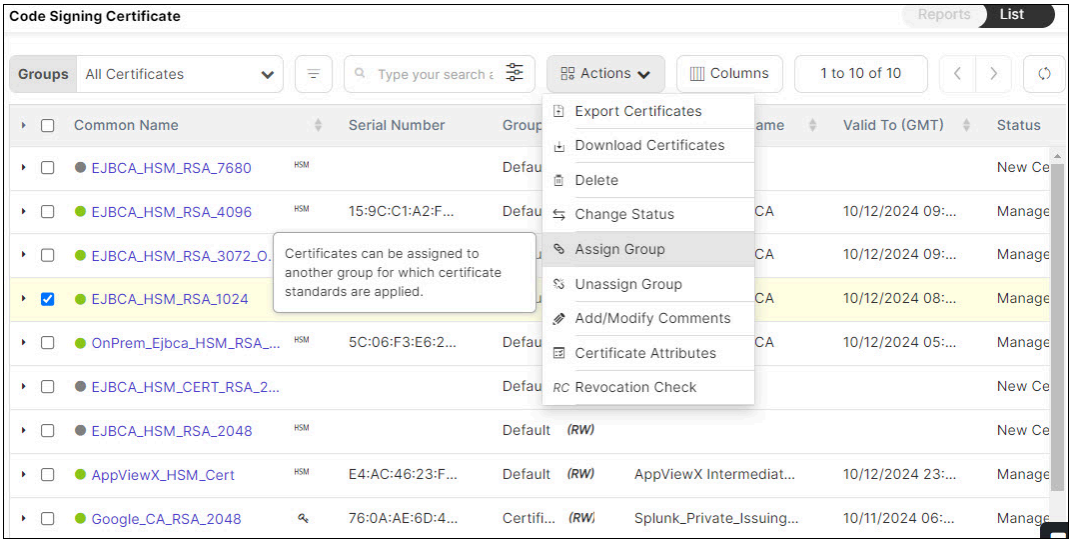
- **Certificate Groups** are used to categorize the certificates according to various **business units**.
- In some organizations, **Certificate Groups** are also used to assign access permissions. Only privileged users (Inherits from Resource > User Group) can view the respective **Certificate Groups**.
- Users should be assigned to a **Role** (Inherited from Role > User Group) that has access to perform the below actions,
  - View a group
  - Assign a group
  - Unassign a group
- With these actions, users can assign a group during **Certificate Discovery** to avoid movement of certificates post-discovery.
- Along with the view, assign, and unassign options, administrators should be assigned to a **Role** that has access for additional actions,
  - Create/modify a group
  - Delete a group
  - Edit Default group.

- Assign Certificate to a Group
- Create a Group
- Modify a Group

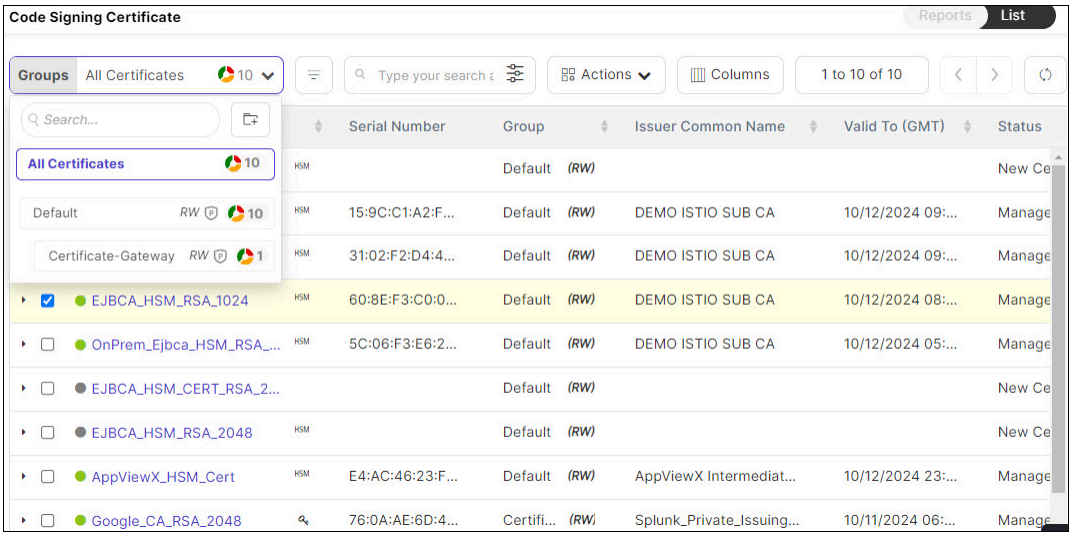
- Delete a Group
- Unassign Certificate from a Group

Assign Certificate to a Group

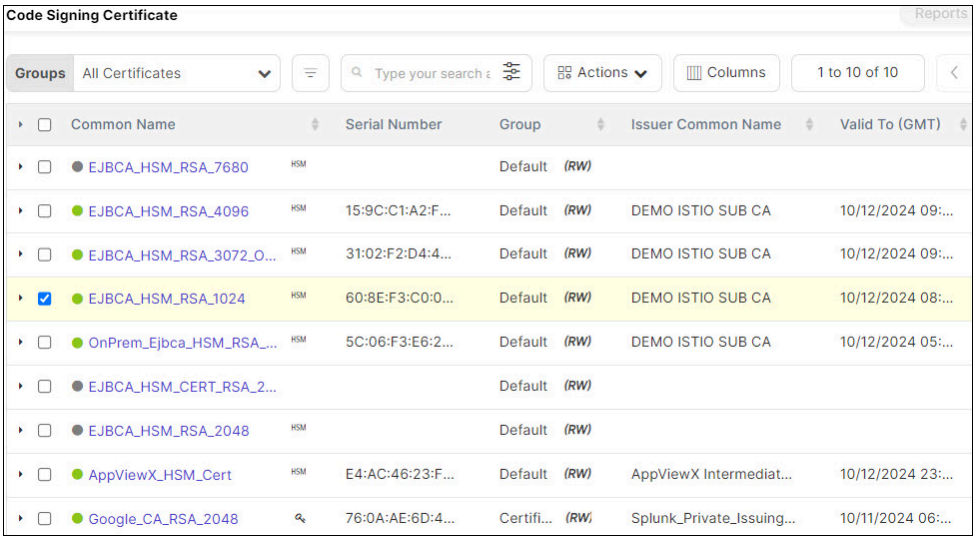
1. Go to **(Menu)** > **SIGN+**.
2. Under the **CERTIFICATE INVENTORY**, select **Code Signing**.
   The **Code Signing Certificate** inventory is displayed.
3. Click **List** button on upper right of the Code Signing Certificate inventory screen.
4. Select the check box against the certificate(s) you want to assign to a group.
5. Click **Actions** drop-down and select the **Assign Group** option from the drop-down.



6. The **Assign to Group** pop-up is displayed. Select the **Group** from the list.
7. Click **Assign** button to move the certificate(s) to the selected **Group**.
8. Click **Groups** drop-down and select your **Group** from the drop-down.



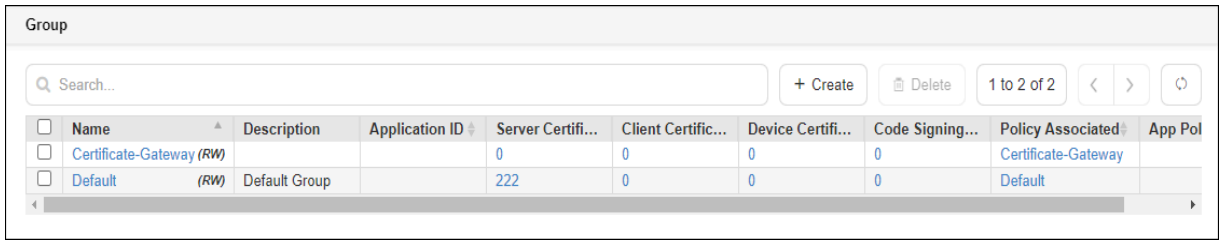
9. You can view the certificate(s) assigned to the **Group**. The table provides certificate(s) details.



Create a Group

Assign the user to a user group that (inherits from resource and role) have access to certificate group.

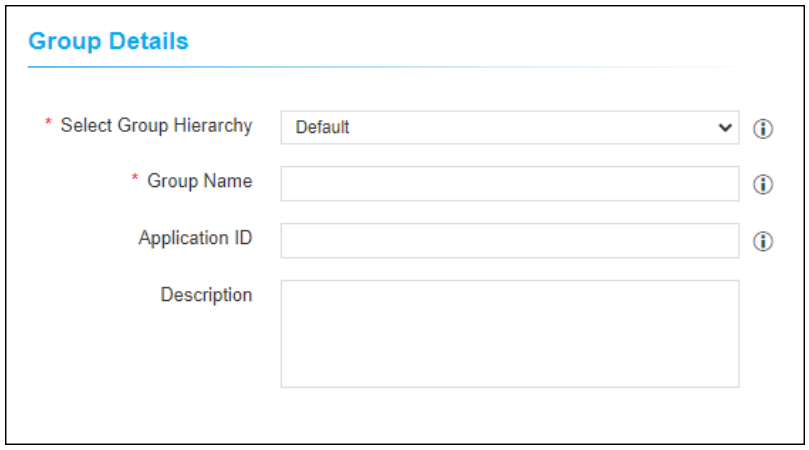
1. Go to **(Menu) > SIGN+ > GROUPS & POLICIES > Groups**.
   The **Group** home page is displayed.
2. **SIGN+** is packaged with default certificate groups **Default** and **Certificate-Gateway**.
3. Click **+ Create** button in the command bar to create a new group.



Field Description for Group Details

Fields	Description
Select Group Hierarchy	Select the parent group to which the new group should be associated
Group Name	Enter a unique name for the new group
Application ID	Provide organization ID (if any) to associate with the new group
Description	Provide the purpose of the new group
*: *Mandatory fields*	

4. Group Name is mandatory in the **Group Details** section. Provide the **Group Name** to create a new group.

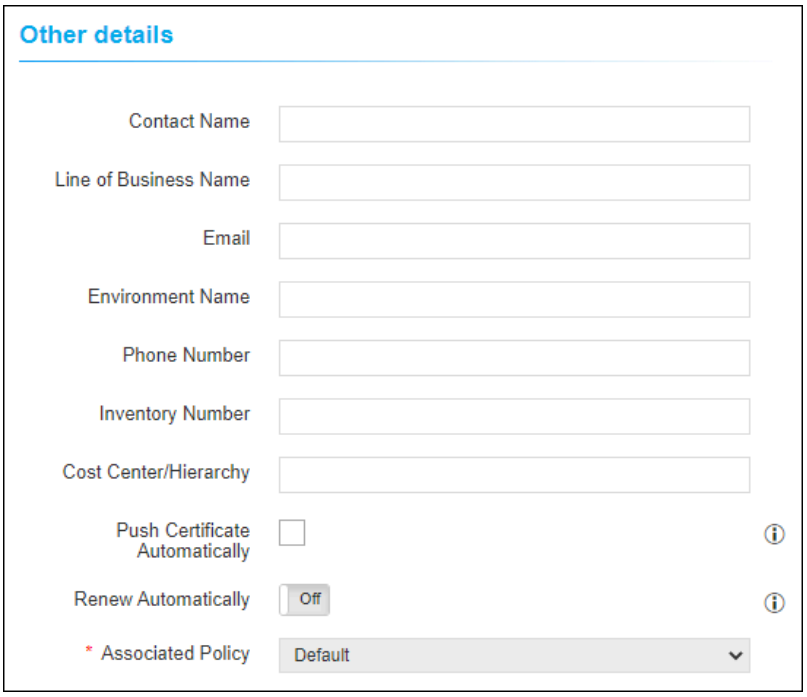


Field Description for Other Details

Fields	Description
Contact Name	Provide contact person to whom changes should be intimated
Line of Business Name	Provide the name of the business unit
Email	Provide contact mail address

Fields	Description
Environment Name	Provide environment name
Phone Number	Provide a phone number for contact
Inventory Number	Provide inventory number
Cost Center/ Hierarchy	Provide Cost Center code/ label
Push Certificate Automatically	By enabling the check box, the renewed/ reissued certificates in this group are automatically associated with their device
Renew Automatically	Turn **On** to automatically renew the certificate belongs to this group.
Associated Policy	Displays the policy associated with this group.
*: *Mandatory fields*	

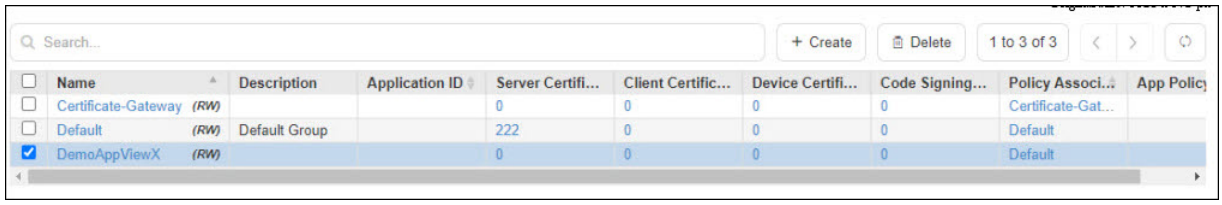
5. The fields in the **Other Details** section are used based on the organization's needs.



6. Click **Create** button to create the group.

   Users can view the group only if it is associated with the **Resource** of their **User Group**. To associate the **Group** with a **Resource**, click the **Update Group and Configure the Resources for User Access** button instead of the **Create** button. This will create the group and navigate to **Resources.** **Refer to** the Create a Resource section of the Platform Guides to configure user access.

7. The newly created **Group** is added to the Group inventory. Click the **Name** (Group name) to view the group details.

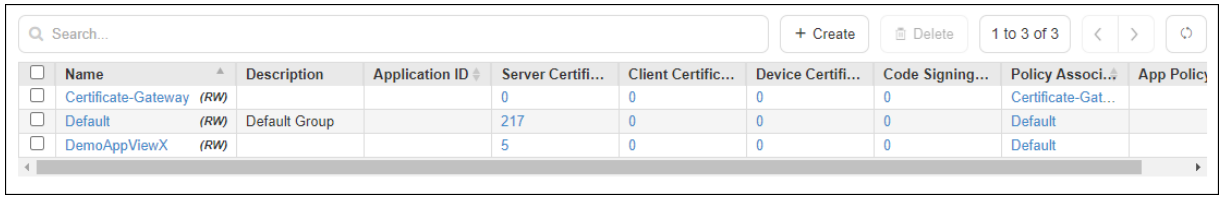


8. Post certificate discovery, you can view the count of certificates Code Signing associated with this group.
9. Click the count in the Server Certificates column to view the certificates.

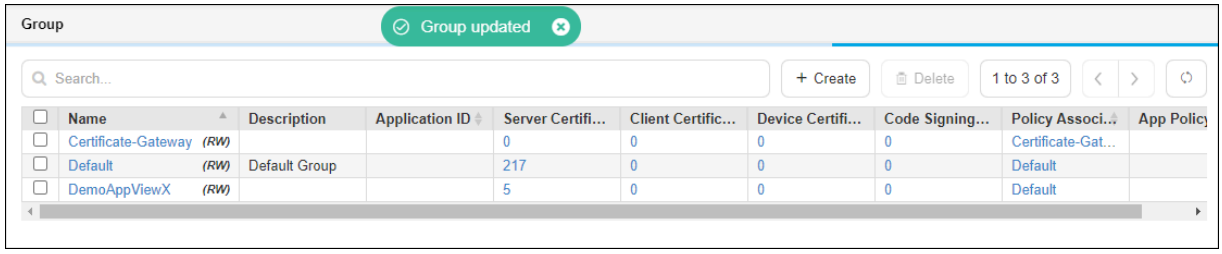
Modify a Group

Assign the user to a user group that (Inherits from resource and role) have access to the certificate group.

1. Go to **(Menu) > SIGN+ > GROUPS & POLICIES > Groups**.
   The **Group** home page is displayed.
2. Click **Name** (Group name) to view the group details.

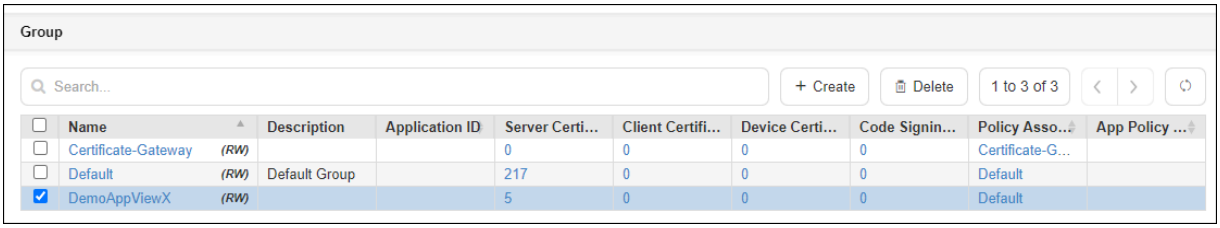


3. Modify required fields in the group and then, click **Update**. Field descriptions are available in Create a Group section.
4. The changes are updated and a confirmation message displays.

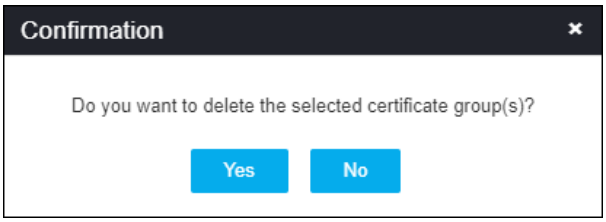


Delete a Group

1. Go to **(Menu) > SIGN+ > GROUPS & POLICIES > Groups**.
   The **Group** home page is displayed.
2. In the group inventory, select the check box against the group you want to delete.
3. Click **(Delete)** icon in the command bar to delete the Group.



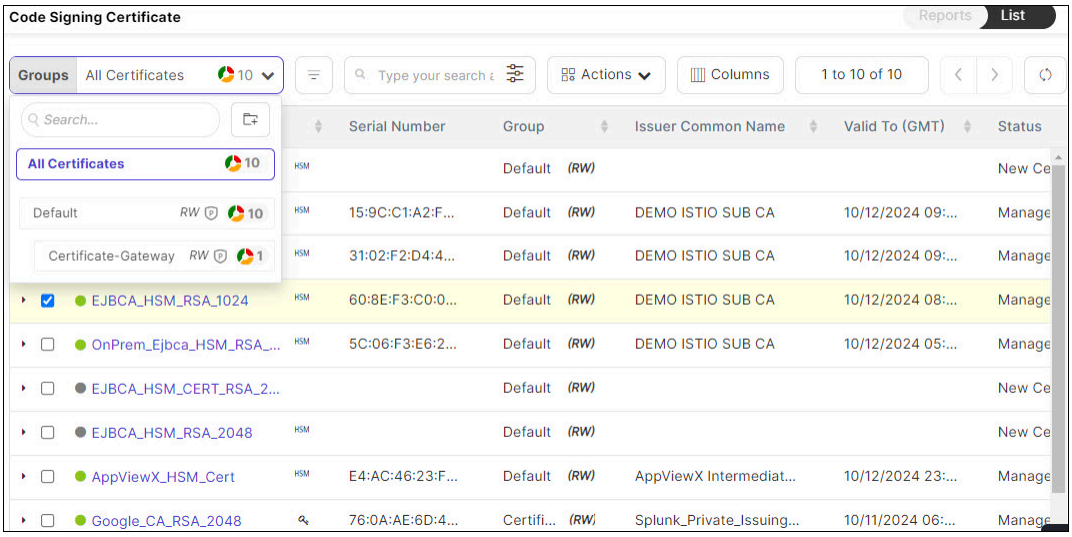
4. A confirmation pop-up is displayed.



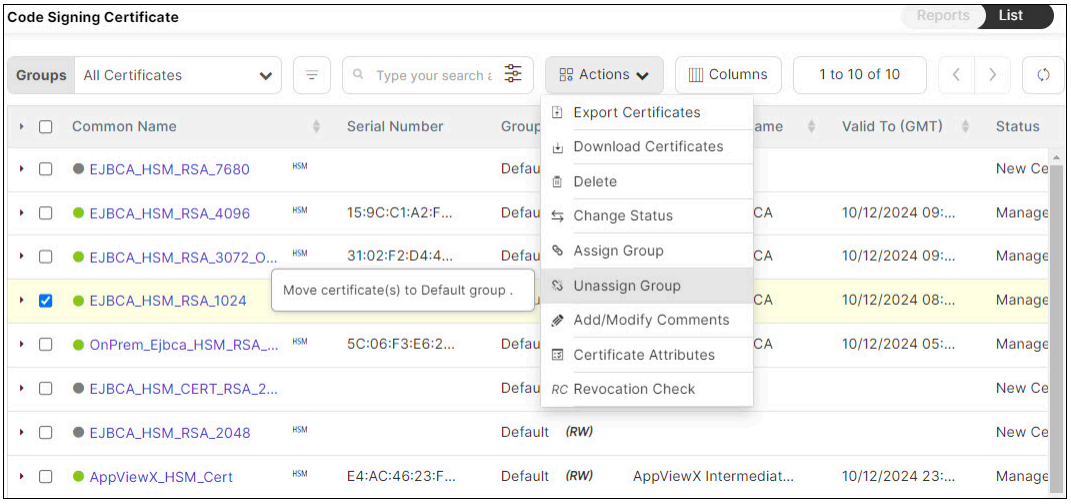
5. Click **Yes** to proceed.
   The group is deleted and a confirmation message displays.

Unassign Certificate from a Group

1. Go to **(Menu)** > **SIGN+**.
2. Under the **CERTIFICATE INVENTORY**, select **Code Signing**.
   The **Code Signing Certificate** inventory is displayed.
3. Click **List** button on the upper right of the Code Signing Certificate inventory screen.
4. Click **Groups** drop-down and select a **Group** from the drop-down.



5. Select the check box against the certificate you want to unassign from the group.
6. Click **Actions** drop-down and select the **Unassign Group** option from the drop-down.



7. The certificate is unassigned from your **Group** and automatically assigned to the **Default Group**.

- Certificates should always be assigned to a group to ensure compliance with the policy.
- When a certificate is unassigned from a group, it will automatically be assigned to the Default Group, ensuring compliance with the Default Policy.

CA Policy

You can enforce your organization standards by configuring a **CA Policy** in **SIGN+**. A CA policy will compare the attributes of discovered certificates against the certificate policy to ensure they are complaint. If the certificate attribute deviates, the certificate is marked non-complaint and thisi is notified

to the users. Users can request the Certificate Authority for a new certificate (in-line to their organization standards).

Prerequisites for configuring a CA policy

- Certificate group(s) must be available to map the policy to them.
- CA accounts (settings) for which a policywill be created must be available.
- Key algorithm, encryption type must be available under the CA accounts.
- AppViewX permission required (**Accounts > Roles** - *Click here to check Accounts management)*

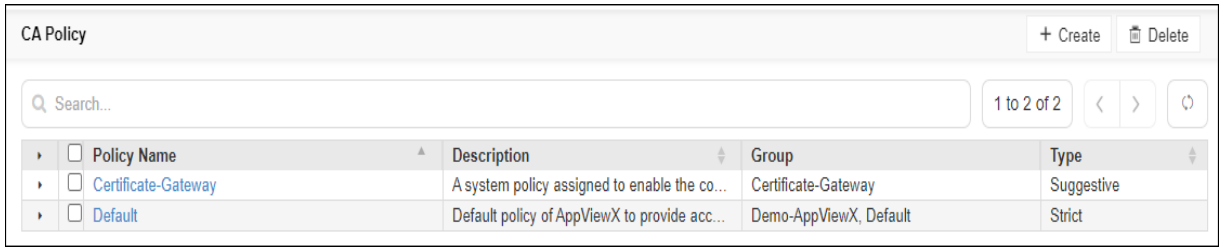
While working with policy

- Go to **SIGN+ > Policy > View Policy** - To view the policy.
- Go to**SIGN+ > Policy > Add / Modify** - To create/ modify the policy.

- Configuring Policy for Symantec CA
- Configuring Policy for Trustwave CA

Configuring Policy Details

1. Go to (**Menu**) > **SIGN+** > **GROUPS & POLICIES** > **CA Policy**.
   The **CA Policy** page is displayed.



> **Note:** SIGN+ is packaged with the following: default policies **Default** and **Certificate-Gateway**.

2. Click **+ Create** from the top-right corner of the page.
   The **CA Policy :: Create** page is displayed.
3. Enter/Select the **Policy Details**.

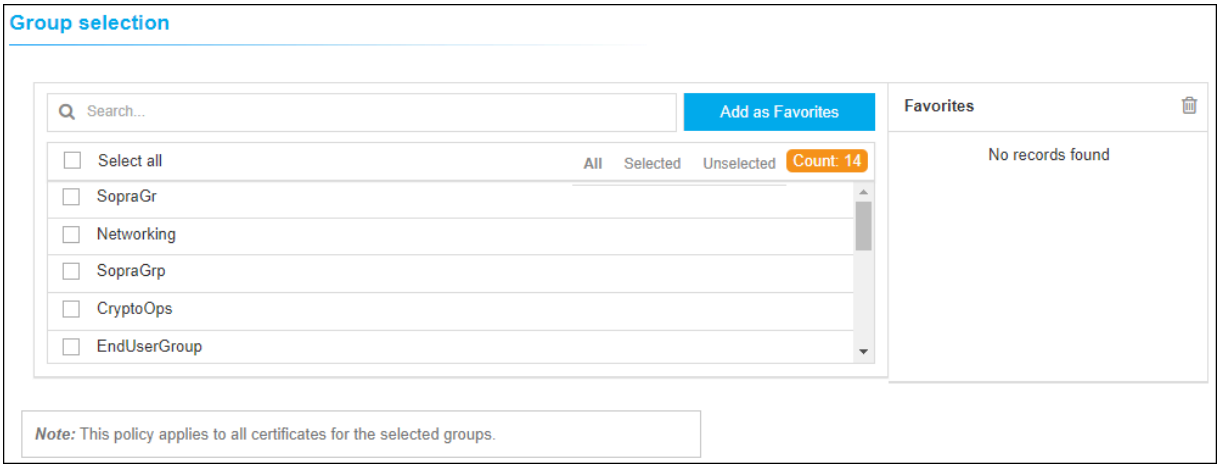
Field description for Policy Details

Fields	Description
***Policy name**	Enter a unique name for the CA policy. **Note:** No special characters other than **.**, **-**, and _ are allowed. The policy name should not start with special characters.
Description	Enter a description of the policy.
***Policy Enforcement Type**	Select **Strict** (default) or **Suggestive.** • **Strict** - Enforces standards defined in the policy where a user cannot modify any parameters. • **Suggestive** - Suggests policy parameters. A user can modify to the suggested values if required.

Fields	Description
Certificate Requests Need Approval	When enabled, this feature will enforce peer approval process for any requests made for creation/renewal/regeneration/reissue or revocation of certificates. Peer approval for requests is defined in the approval workflow.
Enable Access to Private Key	When enabled, allows the user to download private keys from the holistic view.
Enable certificate push-bind access for a read-only user	Enabling this feature will allow a user from a read-only user group to perform certificate push, bind, and rollback operations from the holistic view.
Validate issuer and root certificate for compliance	Enabling this option will validate if the issuer and root of a certificate are also compliant with the standards defined in the policy.
**: Mandatory fields*	

Note: You can configure the **Policy Details** section based on your organization's standards.

4. From the **Group selection**, select one or more groups to map to the policy.



5. From the **Compliance Check** section, to perform an immediate compliance check, enable **Perform Compliance check**.

Note: A scheduled compliance check will run periodically based on the settings defined in the job scheduler.

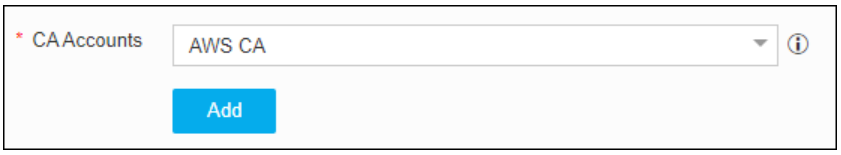
Configuring Policy for Amazon CA

1. Go to (**Menu**) > **SIGN+** > **GROUPS & POLICIES** > **CA Policy**.
   The **CA Policy** page is displayed.
2. Click **+ Create** fromt the top-right of the page.
   The **CA Policy:: Create** page is displayed.
3. Refer the Configuring Policy Details section in the SIGN+ Admin Guide to configure the following:
   - **Policy Details**
   - **Group Selection**
   - **Compliance Check**
4. In the **CA Details** section, from the **Certificate Authority** pane in the left, select **Amazon** .

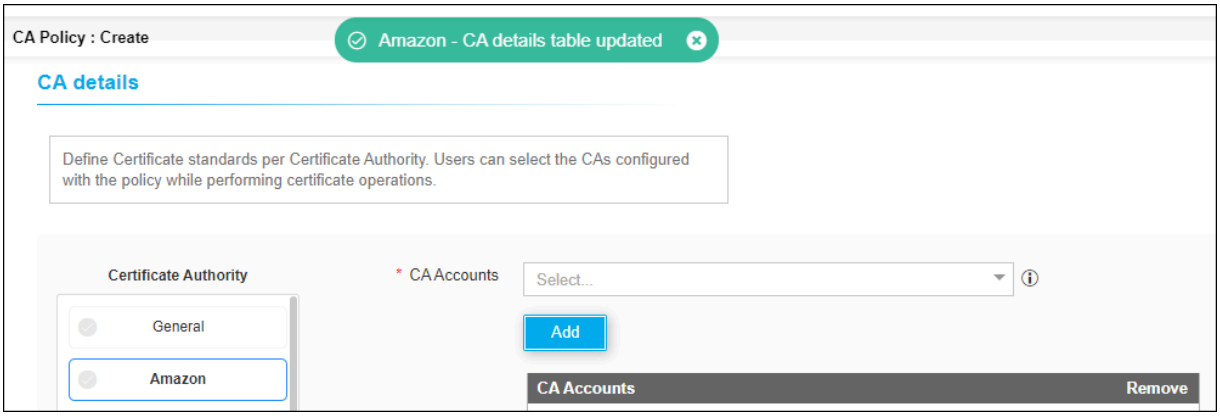
Field description for Amazon CA Details

Fields	Description
***CA Accounts**	The Amazon CA accounts configured in the CA settings screen are listed. Select a CA account from the list to create the policy.
*: *Mandatory fields*	

5. From the **CA Accounts** dropdown list., select the required CA account.



6. Click **Add**.
   The CA details are saved to the table and the confirmation message is displayed.



7. From the ***Bit Length - Key Type** dropdown list, select one (or more than one), bit length- key type pair(s).

The discovered certificate's Key Type and Bit length will be compared against the selected B bit length- key type pair(s) to check for complaince with the policy. The Selected bit length- key type pair(s) is enforced while performing any certificate request operations such as **New**, **Renew**, **Regenerate**.

8. From the ***Hash Function** dropdown list, select one (or more) hash functions.
   The discovered certificate's Key Hash Algorithm will be compared against the selected hash function to check for compliance with the policy. The selected hash function(s) is enforced while performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
9. Enter/Select the **Certificate Parameters**

Field description for certificate parameters

Fields	Description
Host name	Enter the host name. The host name cannot start and end with a **.** (period)
***Allowed Domain Names**	Enter only the white-listed domain names. Press enter after adding the domain name. Multiple domain names can be added.
Common Name	Enter the common name. For example, ***.domain.com** This enforces domains for which a certificate can be requested. The common name is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**. **Note:** Use the * (asterisk) for the host part of the FQDN to enforce the domain. For example, ***.domain.com** will only allow users to request certificates with domain.com. Allowed special characters: ***(Asterisk), **-** (hyphen), **.** (period) .
Subject Alternative Name	Enter the subject alternative name (SAN). It helps enforce additional domains for which a certificate can be requested. The SAN is enforcedat the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.

Fields	Description
	Note: Use the ***** (asterisk) for the host part of the FQDN to enforce the domain. For example, ***.domain.com** will only allow users to request certificates with domain.com. Allowed special characters: *****(Asterisk), **-** (hyphen), **.** (period)
*: *Mandatory fields*	

10. Click the **Save CA Details** button to save the configuration. A green tick mark will be displayed in the **Certificate Authority** pane against the **Amazon** option to indicate that the details are successfully stored.

11. From the **Group selection**, select one or more groups to map to the policy.
12. From the **Compliance Check** section, to perform an immediate compliance check, enable **Perform Compliance check**.

Note: A scheduled compliance check will run periodically based on the settings defined in the job scheduler.

13. Click the **Create Policy** button to create a new policy.
    The policy is created and a confirmation message is displayed.

Configuring Policy for Amazon Private CA

Prerequisites:

- You must configure the CA setting with Amazon Private CA credentials.
- You must have validated and fetched the Amazon Intermediate CAs along with the issuer region details in the CA settings page.

1. Go to (**Menu**) > **SIGN+** > **GROUPS & POLICIES** > **CA Policy**.
   The **CA Policy** page is displayed.
2. Click **+ Create** from the top-right corner of the page.
   The **CA Policy :: Create** page is displayed.

3. Refer the Configuring Policy Details section in the SIGN+ Admin Guide to configure the following:
   - **Policy Details**
   - **Group Selection**
   - **Compliance Check**
4. In the **CA Details** section, from the **Certificate Authority** list in the left, select **Amazon Private CA**. The **CA Details** section is updated to display fields relevant to Amazon Private CA.
5. Enter/Select the policy details for Amazon Private CA.

Field description to create CA policy for Amazon Private CA

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>*CA Accounts</td><td>From the dropdown list, select the certificate authority account.</td></tr>
<tr><td>*Issuer Region</td><td>From the dropdown list, select the issuer region.</td></tr>
<tr><td>*Issuer Name</td><td>From the dropdown list, select the issuer name.</td></tr>
<tr><td>*Validity</td><td>In the Days, Month, and Year dropdown lists, enter the validity period(s) for the certificate.

You can enter more than one validity period in days/months/years, and one can then be chosen from the entered values at the time of certificate enrollment.</td></tr>
<tr><td>*Bit Length - Key Type</td><td>All the key types are listed with their corresponding bit length. You can select one or more than one bit length - key type pair from the dropdown list.

The discovered certificate's key type and bit length will be compared against the selected bit length - key type(s) to to check for compliance with the policy.

The selected bit length - key type(s) is enforced while performing any certificate request operations such as new, renew, regenerate. Amazon Private CA supports the following bit type and length:

<table><tr><th>Type</th><th>Length</th></tr><tr><td>RSA</td><td>2048

4096</td></tr></table></td></tr>
</table>

Field	Description
	<table><tr><th>Type</th><th>Length</th></tr><tr><td>EC</td><td>prime256v1 sec384r1</td></tr></table>
***Hash Function**	From the dropdown list, select one or more than one supported **Hash Function**. The supported hash functions are: • SHA256 • SHA384 • SHA512
***Signature Algorithm**	From the dropdown list, select the required signatire algorithm. **Note:** The issuer will print the issuer algorithm that the users select from the Signature Algorithm in this field.
*: *Mandatory fields*	

6. Enter/Select the **Certificate parameters** values.

Field description for certificate parameters

Field	Description
Restrict Wild Card Certificate	Slide toggle switch to the ON position to restrict the creation of wild card certificates using the policy.
Host name	Enter the host name. The host name cannot start and end with a **.** (period)
***Allowed Domain Names**	Enter only the white-listed domain names. Press enter after adding the domain name. Multiple domain names can be added.
Common Name	Enter the common name. For example, ***.domain.com** This enforces domains for which a certificate can be requested. The common name is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.

Field	Description
	Note: Use the ***** (asterisk) for the host part of the FQDN to enforce the domain. For example, ***.domain.com** will only allow users to request certificates with domain.com. Allowed special characters: *****(Asterisk), **-** (hyphen), **.** (period) .
Organization	Enter the organization name. The discovered certificate's subject organization will be compared against the organization provided in the policy to check for compliance. The organization is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Organization Unit	Enter the organization unit. The discovered certificate's Subject Organization Unit will be compared against the organization unit provided in the policy to check for compliance. Organization Unit is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Locality	Enter the locality name. The discovered certificate's locality will be compared against locality provided in the policy to check for compliance. The locality is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
State	Enter the state. The discovered certificate's state will be compared against the state provided in the policy to check for compliance. The state is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Country code	Enter the country code. The discovered certificate's country code will be compared against the country code provided in the policy to check for compliance. Country code is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Email	Enter the email address of the organization unit.

Field	Description
	The discovered certificate's email address will be compared against the email address provided in the policy to check for compliance. The email address is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Subject Alternative Name	Enter the subject alternative name (SAN). It helps enforce additional domains for which a certificate can be requested. The SAN is enforcedat the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**. **Note:** Use the ***** (asterisk) for the host part of the FQDN to enforce the domain. For example, ***.domain.com** will only allow users to request certificates with domain.com. Allowed special characters: *****(Asterisk), **-** (hyphen), **.** (period)
*: *Mandatory fields*	

7. Click **Save CA Details**.
   A green tick mark is displayed in the **Certificate Authority** pane against **Amazon Private CA** to indicate that the details are successfully stored.
8. From the **Group selection**, select one or more groups to map to the policy.
9. From the **Compliance Check** section, to perform an immediate compliance check, enable **Perform Compliance check**.

   Note: A scheduled compliance check will run periodically based on the settings defined in the job scheduler.

10. Click **Create Policy**.
    The policy is created and a confirmation message is displayed.

Configuring Policy for Digicert CA

1. Go to (**Menu**) > **SIGN+** > **GROUPS & POLICIES** > **CA Policy**.
   The **CA Policy** page is displayed.
2. Click **+ Create** from the top-right corner of the page.

The **CA Policy:: Create** page is displayed.

3. Refer the Configuring Policy Details section in the SIGN+ Admin Guide to configure the following:
   - **Policy Details**
   - **Group Selection**
   - **Compliance Check**
4. In the **CA Details** section, from the **Certificate Authority** list in the left, select **DigiCert**.
   The **CA Details** section is updated to display fields relevant to DigiCert.
5. Enter/Select the policy details for DigiCert.

Field description to create CA policy for DigiCert

Field	Description
***CA Account**	The GlobalSign CA accounts configured in CA settings screen are listed. Select a CA account from the list to create the policy.
***Division**	Select the division from the dropdown list.
***Certificate Type**	Certificate types corresponding to the selected CA account are listed. Select one (or) more certificate types from the list to create the policy.
***Validity**	Enter a validity period for the certificate. The available options are: **Days** - You can enter more than one validity period in days, to choose one in certificate enrolment. **Month** - You can enter more than one validity period in Months, to choose one in certificate enrolment. **Year** - You can enter more than one validity period in Year, to choose one in certificate enrolment.
*: *Mandatory fields*	

6. In the **Vendor Specific Details** section, select/enter the details as listed in the table

Field	Description
***Server Type**	Select the server type from the dropdown list.
*: *Mandatory fields*	

7. Click **Add**.
   The CA details are added to the table below the **Add** button and a confirmation message is displayed.

Note: You can use the **Edit** option in the table to modify the configuration and the **Remove** option to delete the configuration.

CA Accounts	Division	Certificate Type	validity	Edit	Remove
Digicert	private-only	Private SSL Plus	view		
Digicert	public-only	SSL Plus	view		
Digicert	AppViewX Inc.	Private SSL Multi Domain	view		

8. Select the **Bit Length -Key Type**, **ECDSA curve**, and the **Hash Function**.
9. Based on your organization's policies and standards, enter/select values for the **Certificate Parameters**.

Field description for certificate parameters

Field	Description
Restrict Wild Card Certificate	Slide toggle switch to the ON position to restrict the creation of wild card certificates using the policy.
Host name	Enter the host name. The host name cannot start and end with a **.** (period)
***Allowed Domain Names**	Enter only the white-listed domain names. Press enter after adding the domain name. Multiple domain names can be added.
Common Name	Enter the common name. For example, ***.domain.com** This enforces domains for which a certificate can be requested. The common name is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.

Field	Description
	Note: Use the ***** (asterisk) for the host part of the FQDN to enforce the domain. For example, ***.domain.com** will only allow users to request certificates with domain.com. Allowed special characters: *****(Asterisk), **-** (hyphen), **.** (period) .
Organization	Enter the organization name. The discovered certificate's subject organization will be compared against the organization provided in the policy to check for compliance. The organization is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Organization Unit	Enter the organization unit. The discovered certificate's Subject Organization Unit will be compared against the organization unit provided in the policy to check for compliance. Organization Unit is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Locality	Enter the locality name. The discovered certificate's locality will be compared against locality provided in the policy to check for compliance. The locality is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
State	Enter the state. The discovered certificate's state will be compared against the state provided in the policy to check for compliance. The state is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Country code	Enter the country code. The discovered certificate's country code will be compared against the country code provided in the policy to check for compliance. Country code is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Email	Enter the email address of the organization unit.

Field	Description
	The discovered certificate's email address will be compared against the email address provided in the policy to check for compliance. The email address is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Subject Alternative Name	Enter the subject alternative name (SAN). It helps enforce additional domains for which a certificate can be requested. The SAN is enforcedat the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**. **Note:** Use the ***** (asterisk) for the host part of the FQDN to enforce the domain. For example, ***.domain.com** will only allow users to request certificates with domain.com. Allowed special characters: *****(Asterisk), **-** (hyphen), **.** (period)
*: *Mandatory fields*	

10. Click **Save CA Details** to save the configuration.
    A green tick mark is displayed in the **Certificate Authority** pane against **Digicert** to indicate that the details are successfully stored.

11. From the **Group selection**, select one or more groups to map to the policy.
12. From the **Compliance Check** section, to perform an immediate compliance check, enable **Perform Compliance check**.

Note: A scheduled compliance check will run periodically based on the settings defined in the job scheduler.

13. Click **Create Policy** to create a new policy.
    The policy is created and a confirmation message displays.

Configuring Policy for EJBCA CA

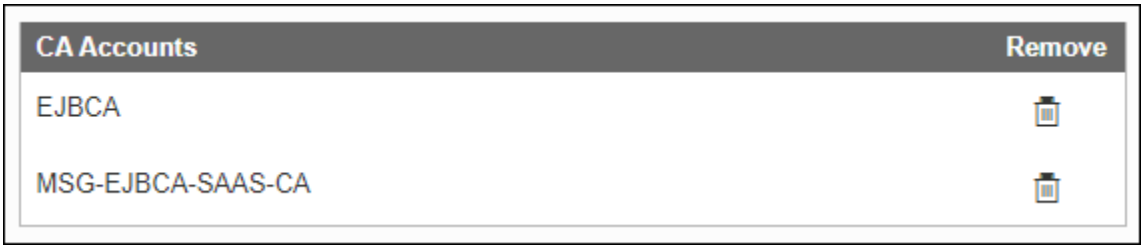
1. Go to (**Menu**) > **SIGN+** > **GROUPS & POLICIES** > **CA Policy**.
   The **CA Policy** page is displayed.
2. Click **+ Create** from the top-right corner of the page.
   The **CA Policy:: Create** page is displayed.
3. Refer the Configuring Policy Details section in the SIGN+ Admin Guide to configure the following:
   - **Policy Details**
   - **Group Selection**
   - **Compliance Check**
4. In the **CA Details** section, from the **Certificate Authority** list in the left, select **EJBCA**.
   The **CA Details** section is updated to display fields relevant to EJBCA.
5. In the **Vendor Specific Details** section, select/enter the details as listed in the table.

Field descriptions for the vendor-specific details

Field	Description
End entity user name	Enter the name of the end entity user.
***End entity Profile name**	Enter the name of the end entity profile.
***Issuer Common Name**	Enter the common user name.
***Certificate Profile Name**	Enter a certificate profile name.
*: *Mandatory fields*	

6. Click **Add** .
   The CA details are saved to the table and the confirmation message is displayed.

   You can use the **Remove** option to delete the configuration.



7. From the ***Bit Length - Key Type** dropdown list, select one (or more than one), bit length- key type pair(s).
   The discovered certificate's Key Type and Bit length will be compared against the selected B
   bit length- key type pair(s) to check for complaince with the policy. The Selected bit length- key

type pair(s) is enforced while performing any certificate request operations such as **New**, **Renew**, **Regenerate**.

8. From the ***Hash Function** dropdown list, select one (or more) hash functions.
   The discovered certificate's Key Hash Algorithm will be compared against the selected hash function to check for compliance with the policy. The selected hash function(s) is enforced while performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
9. Enter/Select the **Certificate parameters** values.

Field description for certificate parameters

Field	Description
Restrict Wild Card Certificate	Slide toggle switch to the ON position to restrict the creation of wild card certificates using the policy.
Host name	Enter the host name. The host name cannot start and end with a **.** (period)
***Allowed Domain Names**	Enter only the white-listed domain names. Press enter after adding the domain name. Multiple domain names can be added.
Common Name	Enter the common name. For example, ***.domain.com** This enforces domains for which a certificate can be requested. The common name is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**. **Note:** Use the ***** (asterisk) for the host part of the FQDN to enforce the domain. For example, ***.domain.com** will only allow users to request certificates with domain.com. Allowed special characters: ***(Asterisk), **-** (hyphen), **.** (period) .
Organization	Enter the organization name. The discovered certificate's subject organization will be compared against the organization provided in the policy to check for compliance. The organization is

Field	Description
	enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Organization Unit	Enter the organization unit. The discovered certificate's Subject Organization Unit will be compared against the organization unit provided in the policy to check for compliance. Organization Unit is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Locality	Enter the locality name. The discovered certificate's locality will be compared against locality provided in the policy to check for compliance. The locality is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
State	Enter the state. The discovered certificate's state will be compared against the state provided in the policy to check for compliance. The state is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Country code	Enter the country code. The discovered certificate's country code will be compared against the country code provided in the policy to check for compliance. Country code is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Email	Enter the email address of the organization unit. The discovered certificate's email address will be compared against the email address provided in the policy to check for compliance. The email address is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Subject Alternative Name	Enter the subject alternative name (SAN). It helps enforce additional domains for which a certificate can be requested. The SAN is enforcedat the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.

Field	Description
	Note: Use the * (asterisk) for the host part of the FQDN to enforce the domain. For example, ***.domain.com** will only allow users to request certificates with domain.com. Allowed special characters: *(Asterisk), - (hyphen), . (period)
*: *Mandatory fields*	

10. Click **Save CA Details** to save the configuration.
    A green tick mark is displayed in the **Certificate Authority** pane against **EJBCA** to indicate the details are successfully stored.

11. From the **Group selection**, select one or more groups to map to the policy.
12. From the **Compliance Check** section, to perform an immediate compliance check, enable **Perform Compliance check**.

Note: A scheduled compliance check will run periodically based on the settings defined in the job scheduler.

13. Click **Create Policy** to create a new policy.
    The policy is created and a confirmation message is displayed.

Configuring Policy for Entrust CA

1. Go to (**Menu**) > **SIGN+** > **GROUPS & POLICIES** > **CA Policy**.
   The **CA Policy** page is displayed.
2. Click **+ Create** from the top-right corner of the page.
   The **CA Policy:: Create** page is displayed.
3. Refer the Configuring Policy Details section in the SIGN+ Admin Guide to configure the following:
   - **Policy Details**
   - **Group Selection**
   - **Compliance Check**
4. In the **CA Details** section, from the **Certificate Authority** list in the left, select **Entrust**.

The **CA Details** section is updated to display fields relevant to Entrust.

5. Enter/Select the CA details for Entrust.

Field Description for CA Details

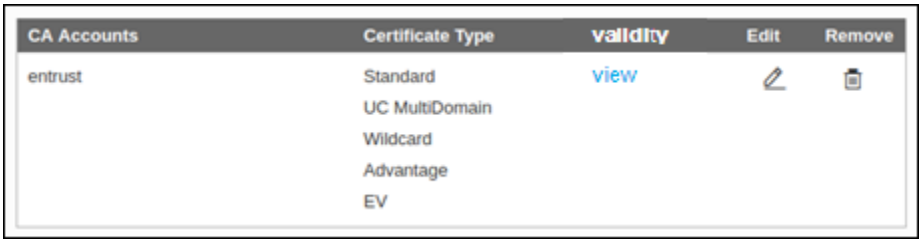
Field	Description
***CA Accounts**	The Entrust CA accounts configured in the CA settings screen are listed. Select a CA account from the list to create the policy.
***Certificate Type**	The **Certificate Types** corresponding to the selected CA account are listed. Select one (or) more certificate types from the list to create the policy.
***Validity**	In the **Days**, **Month**, and **Year** dropdown lists, enter the validity period(s) for the certificate. You can enter more than one validity period in days/months/years, and one can then be chosen from the entered values at the time of certificate enrollment.
*: *Mandatory fields*	

6. In the **Vendor Specific Details** section, select/enter the details as listed in the table:

Field	Description
Additional Emails	Enter the valid email address in the field.

7. Click **Add**.
   The CA details are saved to the table and the confirmation message displays.

   You can use the **Edit** option in the table to modify the configuration and **the Remove** option to delete the configuration.



8. From the ***Bit Length - Key Type** dropdown list, select one (or more than one), bit length- key type pair(s).
   The discovered certificate's Key Type and Bit length will be compared against the selected B
   bit length- key type pair(s) to check for complaince with the policy. The Selected bit length- key

type pair(s) is enforced while performing any certificate request operations such as **New**, **Renew**, **Regenerate**.

9. From the ***Hash Function** dropdown list, select one (or more) hash functions.
   The discovered certificate's Key Hash Algorithm will be compared against the selected hash function to check for compliance with the policy. The selected hash function(s) is enforced while performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
10. Enter/Select the **Certificate parameters** values.

Field description for certificate parameters

Field	Description
Common Name	Enter the common name. For example, ***.domain.com** This enforces domains for which a certificate can be requested. The common name is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**. **Note:** Use the * (asterisk) for the host part of the FQDN to enforce the domain. For example, ***.domain.com** will only allow users to request certificates with domain.com. Allowed special characters: *(Asterisk), **-** (hyphen), **.** (period) .
Organization	Enter the organization name. The discovered certificate's subject organization will be compared against the organization provided in the policy to check for compliance. The organization is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Organization Unit	Enter the organization unit. The discovered certificate's Subject Organization Unit will be compared against the organization unit provided in the policy to check for compliance. Organization Unit is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Locality	Enter the locality name.

Field	Description
	The discovered certificate's locality will be compared against locality provided in the policy to check for compliance. The locality is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
State	Enter the state. The discovered certificate's state will be compared against the state provided in the policy to check for compliance. The state is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Country code	Enter the country code. The discovered certificate's country code will be compared against the country code provided in the policy to check for compliance. Country code is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Email	Enter the email address of the organization unit. The discovered certificate's email address will be compared against the email address provided in the policy to check for compliance. The email address is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Subject Alternative Name	Enter the subject alternative name (SAN). It helps enforce additional domains for which a certificate can be requested. The SAN is enforcedat the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**. **Note:** Use the ***** (asterisk) for the host part of the FQDN to enforce the domain. For example, ***.domain.com** will only allow users to request certificates with domain.com. Allowed special characters: *****(Asterisk), **-** (hyphen), **.** (period)
*: *Mandatory fields*	

11. Click **Save CA Details** to save the configuration.
    A green tick mark will be displayed in the **Certificate Authority** pane against **Entrust** option to indicate that the details are successfully stored.

12. From the **Group selection**, select one or more groups to map to the policy.
13. From the **Compliance Check** section, to perform an immediate compliance check, enable **Perform Compliance check**.

> **Note:** A scheduled compliance check will run periodically based on the settings defined in the job scheduler.

14. Click **Create Policy** to create a new policy.
    The policy is created and a confirmation message is displayed.

Configuring Policy for Entrust MPKI CA

1. Go to (**Menu**) > **SIGN+** > **GROUPS & POLICIES** > **CA Policy**.
   The **CA Policy** page is displayed.
2. Click **+ Create** from the top-right corner of the page.
   The **CA Policy:: Create** page is displayed.
3. Refer the Configuring Policy Details section in the SIGN+ Admin Guide to configure the following:
   - **Policy Details**
   - **Group Selection**
   - **Compliance Check**
4. In the **CA Details** section, from the **Certificate Authority** list in the left, select **Entrust MPKI**.
   The **CA Details** section is updated to display fields relevant to Amazon Private CA.
5. Enter/Select the policy details for Entrust MPKI.

Field Description for CA Details

Field	Description
***CA Accounts**	Select a CA account from the list to create the policy. The selected CA account will be listed in the **CA Accounts** table.
***Bit Length - Key Type**	From the dropdown list, select one (or more than one), bit length- key type pair(s). The discovered certificate's Key Type and Bit length will be compared against the selected B bit length- key type pair(s) to check for complaince with the policy. The Selected bit length- key type pair(s) is enforced while performing any certificate request operations such as **New**, **Renew**, **Regenerate**.

Field	Description
***Hash Function**	From the dropdown list, select one (or more) hash functions. The discovered certificate's Key Hash Algorithm will be compared against the selected hash function to check for compliance with the policy. The selected hash function(s) is enforced while performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
*: *Mandatory fields*	

6. Enter/Select the **Certificate parameters** values.

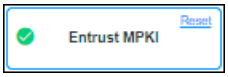
Field description for certificate parameters

Field	Description
Host name	Enter the host name. The host name cannot start and end with a **.** (period)
***Allowed Domain Names**	Enter only the white-listed domain names. Press enter after adding the domain name. Multiple domain names can be added.
Common Name	Enter the common name. For example, ***.domain.com** This enforces domains for which a certificate can be requested. The common name is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**. **Note:** Use the * (asterisk) for the host part of the FQDN to enforce the domain. For example, ***.domain.com** will only allow users to request certificates with domain.com. Allowed special characters: *(Asterisk), **-** (hyphen), **.** (period) .
Organization	Enter the organization name. The discovered certificate's subject organization will be compared against the organization provided in the policy to check for compliance. The organization is

Field	Description
	enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Organization Unit	Enter the organization unit. The discovered certificate's Subject Organization Unit will be compared against the organization unit provided in the policy to check for compliance. Organization Unit is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Locality	Enter the locality name. The discovered certificate's locality will be compared against locality provided in the policy to check for compliance. The locality is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
State	Enter the state. The discovered certificate's state will be compared against the state provided in the policy to check for compliance. The state is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Country code	Enter the country code. The discovered certificate's country code will be compared against the country code provided in the policy to check for compliance. Country code is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Email	Enter the email address of the organization unit. The discovered certificate's email address will be compared against the email address provided in the policy to check for compliance. The email address is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Subject Alternative Name	Enter the subject alternative name (SAN). It helps enforce additional domains for which a certificate can be requested. The SAN is enforcedat the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.

Field	Description
	Note: Use the * (asterisk) for the host part of the FQDN to enforce the domain. For example, ***.domain.com** will only allow users to request certificates with domain.com. Allowed special characters: *(Asterisk), - (hyphen), . (period)
*: *Mandatory fields*	

7. Click **Save CA Details** to save the configuration.
   A green tick mark is displayed in the **Certificate Authority** pane against **Entrust MPKI** to indicate that the details are successfully stored.



8. From the **Group selection**, select one or more groups to map to the policy.
9. From the **Compliance Check** section, to perform an immediate compliance check, enable **Perform Compliance check**.

> **Note:** A scheduled compliance check will run periodically based on the settings defined in the job scheduler.

10. Click **Create Policy** to create a new policy.
    The policy is created and a confirmation message is displayed.

Configuring Policy for GlobalSign CA

1. Go to (**Menu**) > **SIGN+** > **GROUPS & POLICIES** > **CA Policy**.
   The **CA Policy** page is displayed.
2. Click **+ Create** from the top-right corner of the page.
   The **CA Policy: Create** page is displayed.
3. Refer the Configuring Policy Details section in the SIGN+ Admin Guide to configure the following:
   - **Policy Details**
   - **Group Selection**
   - **Compliance Check**
4. In the **CA Details** section, from the **Certificate Authority** list in the left, select **EJBCA**.

The **CA Details** section is updated to display fields relevant to EJBCA.

5. In the **Vendor Specific Details** section, select/enter the details as listed in the table.

Field	Description
***Incorporating Agency Reg. No**	Enter the agency registration number.
***Designation**	Enter the designation.
***Business Category**	Select the business category from the dropdown list.
*: *Mandatory fields*	

6. Click **Add**.
   The CA details are saved to the table and the confirmation message is displayed.

   You can use the **Edit** option in the table to modify the configuration and **Remove** option to delete the configuration.

7. From the ***Bit Length - Key Type** dropdown list, select one (or more than one), bit length- key type pair(s).
   The discovered certificate's Key Type and Bit length will be compared against the selected B bit length- key type pair(s) to check for complaince with the policy. The Selected bit length- key type pair(s) is enforced while performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
8. From the ***Hash Function** dropdown list, select one (or more) hash functions.
   The discovered certificate's Key Hash Algorithm will be compared against the selected hash function to check for compliance with the policy. The selected hash function(s) is enforced while performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
9. Enter/Select the **Certificate parameters** values.

Field description for certificate parameters

Field	Description
Restrict Wild Card Certificate	Slide toggle switch to the ON position to restrict the creation of wild card certificates using the policy.
Host name	Enter the host name. The host name cannot start and end with a **.** (period)

Field	Description
***Allowed Domain Names**	Enter only the white-listed domain names. Press enter after adding the domain name. Multiple domain names can be added.
Common Name	Enter the common name. For example, ***.domain.com** This enforces domains for which a certificate can be requested. The common name is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**. **Note:** Use the * (asterisk) for the host part of the FQDN to enforce the domain. For example, ***.domain.com** will only allow users to request certificates with domain.com. Allowed special characters: *(Asterisk), **-** (hyphen), **.** (period) .
Organization	Enter the organization name. The discovered certificate's subject organization will be compared against the organization provided in the policy to check for compliance. The organization is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Organization Unit	Enter the organization unit. The discovered certificate's Subject Organization Unit will be compared against the organization unit provided in the policy to check for compliance. Organization Unit is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Locality	Enter the locality name. The discovered certificate's locality will be compared against locality provided in the policy to check for compliance. The locality is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
State	Enter the state.

Field	Description
	The discovered certificate's state will be compared against the state provided in the policy to check for compliance. The state is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Country code	Enter the country code. The discovered certificate's country code will be compared against the country code provided in the policy to check for compliance. Country code is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Email	Enter the email address of the organization unit. The discovered certificate's email address will be compared against the email address provided in the policy to check for compliance. The email address is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Subject Alternative Name	Enter the subject alternative name (SAN). It helps enforce additional domains for which a certificate can be requested. The SAN is enforcedat the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**. **Note:** Use the ***** (asterisk) for the host part of the FQDN to enforce the domain. For example, ***.domain.com** will only allow users to request certificates with domain.com. Allowed special characters: *****(Asterisk), **-** (hyphen), **.** (period)
*: *Mandatory fields*	

10. Click **Save CA Details** to save the configuration.
    A green tick mark is displayed in the **Certificate Authority** pane against **GlobalSign** to indicate that the details are successfully stored.
11. From the **Group selection**, select one or more groups to map to the policy.
12. From the **Compliance Check** section, to perform an immediate compliance check, enable **Perform Compliance check**.

Note: A scheduled compliance check will run periodically based on the settings defined in the job scheduler.

13. Click **Create Policy** to create a new policy.
    The policy is created and a confirmation message is displayed.

Configuring Policy for GlobalSign MSSL CA

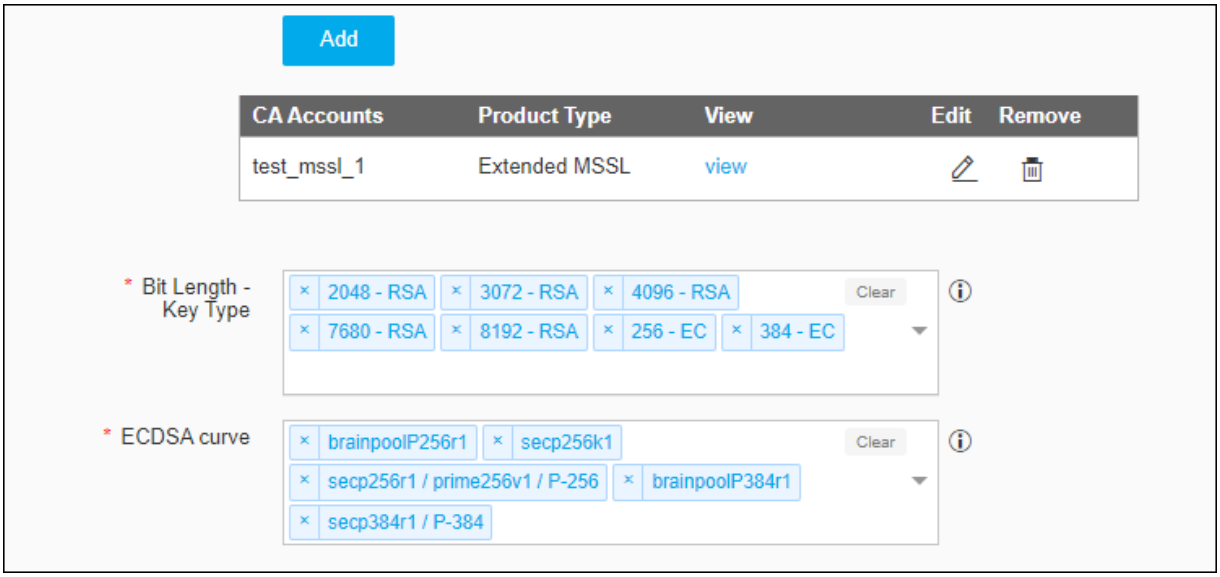
1. Go to (**Menu**) > **SIGN+** > **GROUPS & POLICIES** > **CA Policy**.
   The **CA Policy** page is displayed.
2. Click **+ Create** to configure a GlobalSign MSSL based policy.
   The **CA Policy:: Create** page is displayed.
3. Refer the Configuring Policy Details section in the SIGN+ Admin Guide to configure the following:
   - **Policy Details**
   - **Group Selection**
   - **Compliance Check**
4. In the **CA Details** section, from the **Certificate Authority** list in the left, select **GlobalSign MSSL**.
   The **CA Details** section is updated to display fields relevant to GlobalSign MSSL.
5. Enter/Select the CA details.

Field Description for CA Details

Field	Description
***CA Accounts**	The GlobalSign MSSL CA accounts configured on the CA settings screen are listed. Select a CA account from the list to create the policy.
* **Product Type**	All Managed SSL Product Types require that the Organization's information and at least one Domain be registered in the Managed SSL account prior to ordering.
***Signature Algorithm**	Select the signature algorithm from the drop-down list.
* **MSSL Profile Allowed Domain Name**	Select the **MSSL Profile Allowed Domain Name** from the drop-down list.

Field	Description
***Validity**	In the **Days**, **Month**, and **Year** dropdown lists, enter the validity period(s) for the certificate. You can enter more than one validity period in days/months/years, and one can then be chosen from the entered values at the time of certificate enrollment.
*: *Mandatory fields*	

6. Click **Add**.
   The CA details are saved to the table and the confirmation message is displayed.



   You can use **Edit** option in the table to modify the configuration and the **Remove** option to delete the configuration.

7. From the ***Bit Length - Key Type** dropdown list, select one (or more than one), bit length- key type pair(s).
   The discovered certificate's Key Type and Bit length will be compared against the selected B bit length- key type pair(s) to check for complaince with the policy. The Selected bit length- key type pair(s) is enforced while performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
8. From the ***Hash Function** dropdown list, select one (or more) hash functions.

The discovered certificate's Key Hash Algorithm will be compared against the selected hash function to check for compliance with the policy. The selected hash function(s) is enforced while performing any certificate request operations such as **New**, **Renew**, **Regenerate**.

9. Enter/Select the **Certificate parameters** values.

Field description for certificate parameters

Field	Description
Restrict Wild Card Certificate	Slide toggle switch to the ON position to restrict the creation of wild card certificates using the policy.
Host name	Enter the host name. The host name cannot start and end with a **.** (period)
***Allowed Domain Names**	Enter only the white-listed domain names. Press enter after adding the domain name. Multiple domain names can be added.
Common Name	Enter the common name. For example, ***.domain.com** This enforces domains for which a certificate can be requested. The common name is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**. **Note:** Use the ***** (asterisk) for the host part of the FQDN to enforce the domain. For example, ***.domain.com** will only allow users to request certificates with domain.com. Allowed special characters: ***(Asterisk), **-** (hyphen), **.** (period) .
Organization	Enter the organization name. The discovered certificate's subject organization will be compared against the organization provided in the policy to check for compliance. The organization is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Organization Unit	Enter the organization unit. The discovered certificate's Subject Organization Unit will be compared against the organization unit provided in the policy to check for

Field	Description
	compliance. Organization Unit is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Locality	Enter the locality name. The discovered certificate's locality will be compared against locality provided in the policy to check for compliance. The locality is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
State	Enter the state. The discovered certificate's state will be compared against the state provided in the policy to check for compliance. The state is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Country code	Enter the country code. The discovered certificate's country code will be compared against the country code provided in the policy to check for compliance. Country code is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Email	Enter the email address of the organization unit. The discovered certificate's email address will be compared against the email address provided in the policy to check for compliance. The email address is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Subject Alternative Name	Enter the subject alternative name (SAN). It helps enforce additional domains for which a certificate can be requested. The SAN is enforcedat the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**. **Note:** Use the ***** (asterisk) for the host part of the FQDN to enforce the domain. For example, ***.domain.com** will only allow users to request certificates with domain.com. Allowed special characters: *****(Asterisk), **-** (hyphen), **.** (period)
*: *Mandatory fields*	

10. Click **Save CA Details** to save the configuration.
    A green tick mark is displayed in the **Certificate Authority** pane against **GlobalSign MSSL** to indicate that the details are successfully stored.
11. From the **Group selection**, select one or more groups to map to the policy.
12. From the **Compliance Check** section, to perform an immediate compliance check, enable **Perform Compliance check**.

    Note: A scheduled compliance check will run periodically based on the settings defined in the job scheduler.

13. Click **Create Policy** to create a new policy.
    The policy is created and a confirmation message is displayed.

Configuring Policy for GlobalSign Atlas CA

Before You Begin: The prerequisites for configuring the policy are as follows:

- Certificate group(s) must be available to map the policy to them
- CA accounts (settings) must be available to which the policy is going to be created
- AppViewX permission required (**Accounts** > **Roles** - Click here to check Accounts management)

To configure policy for GlobalSign Atlas CA:

1. Go to (**Menu**) > **SIGN+** > **GROUPS & POLICIES** > **CA Policy**.
   The **CA Policy** page is displayed.

   SIGN+ is packaged with the following default policies: **Default** and **Certificate-Gateway**.

   Note: The **Default** CA Policy will have the **GlobalSign Atlas CA** details.

2. Click **+ Create** to configure GlobalSign Atlas custom policy.
   The **CA Policy:: Create** page is displayed.
3. Refer the Configuring Policy Details section in the SIGN+ Admin Guide to configure the following:
   - **Policy Details**
   - **Group Selection**
   - **Compliance Check**
4. In the **CA Details** section, from the **Certificate Authority** list in the left, select **GlobalSign Atlas**.
   The **CA Details** section is updated to display fields relevant to GlobalSign Atlas.

5. Enter/Select the CA details.

Field description for CA details

Field	Description
***CA Accounts**	The GlobalSign Atlas CA accounts configured in the CA settings screen are listed here. Select a CA account from the list to create the policy.
***Validity**	In the **Days**, **Month**, and **Year** dropdown lists, enter the validity period(s) for the certificate. You can enter more than one validity period in days/months/years, and one can then be chosen from the entered values at the time of certificate enrollment.
***Bit Length - Key Type**	From the dropdown list, select one (or more than one), bit length- key type pair(s). The discovered certificate's Key Type and Bit length will be compared against the selected B bit length- key type pair(s) to check for complaince with the policy. The Selected bit length- key type pair(s) is enforced while performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
***Hash Function**	From the dropdown list, select one (or more) hash functions. The discovered certificate's Key Hash Algorithm will be compared against the selected hash function to check for compliance with the policy. The selected hash function(s) is enforced while performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
*: *Mandatory fields*	

The updated fields for the CA are displayed on the right.

6. Click **Add**.
   The CA details are saved in the table and the confirmation message is displayed.

   You can use the **Edit** (pencil)option in the table to modify the configuration and the **Remove** (bin) option to delete the configuration.

7. Enter/Select the **Certificate parameters** values.

Field description for certificate parameters

Field	Description
Restrict Wild Card Certificate	Slide toggle switch to the ON position to restrict the creation of wild card certificates using the policy.
Host name	Enter the host name. The host name cannot start and end with a **.** (period)
***Allowed Domain Names**	Enter only the white-listed domain names. Press enter after adding the domain name. Multiple domain names can be added.
Common Name	Enter the common name. For example, ***.domain.com** This enforces domains for which a certificate can be requested. The common name is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**. **Note:** Use the * (asterisk) for the host part of the FQDN to enforce the domain. For example, ***.domain.com** will only allow users to request certificates with domain.com. Allowed special characters: *(Asterisk), **-** (hyphen), **.** (period) .
Organization	Enter the organization name. The discovered certificate's subject organization will be compared against the organization provided in the policy to check for compliance. The organization is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Organization Unit	Enter the organization unit. The discovered certificate's Subject Organization Unit will be compared against the organization unit provided in the policy to check for compliance. Organization Unit is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Locality	Enter the locality name.

Field	Description
	The discovered certificate's locality will be compared against locality provided in the policy to check for compliance. The locality is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
State	Enter the state. The discovered certificate's state will be compared against the state provided in the policy to check for compliance. The state is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Country code	Enter the country code. The discovered certificate's country code will be compared against the country code provided in the policy to check for compliance. Country code is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Email	Enter the email address of the organization unit. The discovered certificate's email address will be compared against the email address provided in the policy to check for compliance. The email address is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Subject Alternative Name	Enter the subject alternative name (SAN). It helps enforce additional domains for which a certificate can be requested. The SAN is enforcedat the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**. **Note:** Use the ***** (asterisk) for the host part of the FQDN to enforce the domain. For example, ***.domain.com** will only allow users to request certificates with domain.com. Allowed special characters: *****(Asterisk), **-** (hyphen), **.** (period)
*: *Mandatory fields*	

8. Click **Save CA Details** to save the configuration.
   A green tick mark is displayed in the **Certificate Authority** pane against **GlobalSign MSSL** to indicate that the details are successfully stored.

9. From the **Group selection**, select one or more groups to map to the policy.
10. From the **Compliance Check** section, to perform an immediate compliance check, enable **Perform Compliance check**.

> **Note:** A scheduled compliance check will run periodically based on the settings defined in the job scheduler.

11. Click **Create Policy** button to create a new policy.
    The policy is created and a confirmation message is displayed.

Configuring Policy for GoDaddy CA

1. Go to (**Menu**) > **SIGN+** > **GROUPS & POLICIES** > **CA Policy**.
   The **CA Policy** page is displayed.
2. Click **+ Create** from the top-right corner of the page.
   The **CA Policy:: Create** page is displayed.
3. Refer the Configuring Policy Details section in the SIGN+ Admin Guide to configure the following:
   - **Policy Details**
   - **Group Selection**
   - **Compliance Check**
4. In the **CA Details** section, from the **Certificate Authority** list in the left, select **GoDaddy**.
   The **CA Details** section is updated to display fields relevant to GoDaddy.
5. Enter/Select the CA details.

Field description for CA details

Field	Description
***CA Account**	The GoDaddy CA accounts configured in CA settings screen are listed. Select a CA account from the list to create the policy.
***Certificate Type**	The **Certificate Types** corresponding to the selected CA account are listed. Select one (or) more **Certificate Type** from the list to create the policy.
***Validity**	In the **Days**, **Month**, and **Year** dropdown lists, enter the validity period(s) for the certificate.

Field	Description
	You can enter more than one validity period in days/months/years, and one can then be chosen from the entered values at the time of certificate enrollment.
*: *Mandatory fields*	

6. From the ***Bit Length - Key Type** dropdown list, select one (or more than one), bit length- key type pair(s).
   The discovered certificate's Key Type and Bit length will be compared against the selected B bit length- key type pair(s) to check for complaince with the policy. The Selected bit length- key type pair(s) is enforced while performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
7. From the ***Hash Function** dropdown list, select one (or more) hash functions.
   The discovered certificate's Key Hash Algorithm will be compared against the selected hash function to check for compliance with the policy. The selected hash function(s) is enforced while performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
8. Click **Add**.
   The CA details are saved to the table and the confirmation message is displayed.

   You can use **Edit** option in the table to modify the configuration and **Remove** option to delete the configuration.
9. Enter/Select the certificate parameters.

Field description for certificate parameters

Field	Description
Restrict Wild Card Certificate	Slide toggle switch to the ON position to restrict the creation of wild card certificates using the policy.
Host name	Enter the host name. The host name cannot start and end with a **.** (period)
***Allowed Domain Names**	Enter only the white-listed domain names. Press enter after adding the domain name. Multiple domain names can be added.

Field	Description
Common Name	Enter the common name. For example, ***.domain.com** This enforces domains for which a certificate can be requested. The common name is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**. **Note:** Use the * (asterisk) for the host part of the FQDN to enforce the domain. For example, ***.domain.com** will only allow users to request certificates with domain.com. Allowed special characters: *(Asterisk), **-** (hyphen), **.** (period) .
Organization	Enter the organization name. The discovered certificate's subject organization will be compared against the organization provided in the policy to check for compliance. The organization is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Organization Unit	Enter the organization unit. The discovered certificate's Subject Organization Unit will be compared against the organization unit provided in the policy to check for compliance. Organization Unit is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Locality	Enter the locality name. The discovered certificate's locality will be compared against locality provided in the policy to check for compliance. The locality is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
State	Enter the state. The discovered certificate's state will be compared against the state provided in the policy to check for compliance. The state is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Country code	Enter the country code.

Field	Description
	The discovered certificate's country code will be compared against the country code provided in the policy to check for compliance. Country code is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Email	Enter the email address of the organization unit. The discovered certificate's email address will be compared against the email address provided in the policy to check for compliance. The email address is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Subject Alternative Name	Enter the subject alternative name (SAN). It helps enforce additional domains for which a certificate can be requested. The SAN is enforcedat the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**. **Note:** Use the ***** (asterisk) for the host part of the FQDN to enforce the domain. For example, ***.domain.com** will only allow users to request certificates with domain.com. Allowed special characters: *****(Asterisk), **-** (hyphen), **.** (period)
**: Mandatory fields*	

10. Click **Save CA Details** to save the configuration.
    A green tick mark is displayed in the **Certificate Authority** pane against **GoDaddy** to indicate that the details are successfully stored.

11. From the **Group selection**, select one or more groups to map to the policy.
12. From the **Compliance Check** section, to perform an immediate compliance check, enable **Perform Compliance check**.

Note: A scheduled compliance check will run periodically based on the settings defined in the job scheduler.

13. Click **Create Policy** button to create a new policy.
    The policy is created and a confirmation message is displayed.

Configuring Policy for Google CA

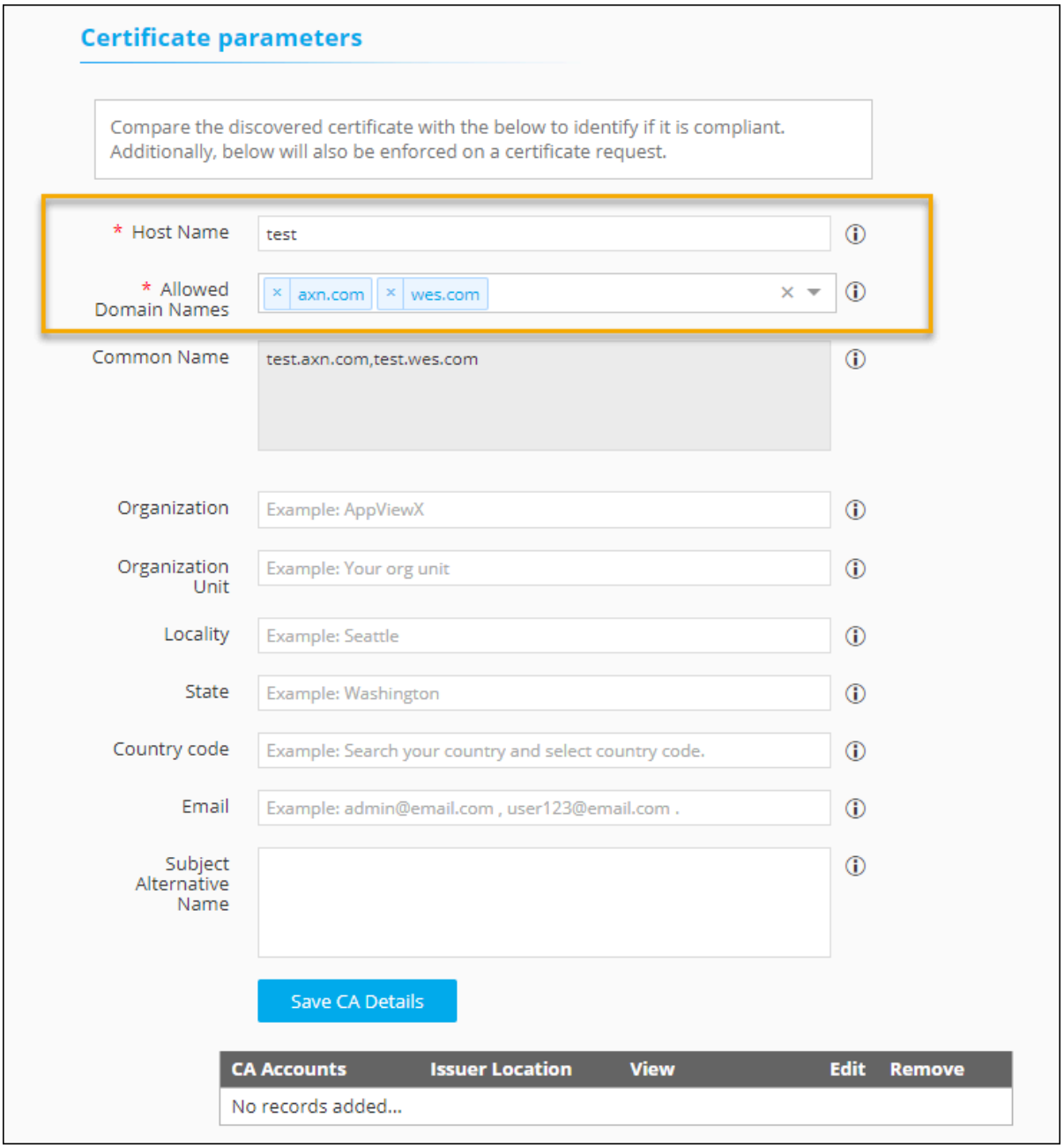
1. Go to (**Menu**) > **SIGN+** > **GROUPS & POLICIES** > **CA Policy**.
   The **CA Policy** page is displayed.
2. Click **+ Create** from the top-right corner of the page.
   The **CA Policy: Create** page is displayed.
3. Refer the Configuring Policy Details section in the SIGN+ Admin Guide to configure the following:
   - **Policy Details**
   - **Group Selection**
   - **Compliance Check**
4. In the **CA Details** section, from the **Certificate Authority** list in the left, select **Google**.
   The **CA Details** section is updated to display fields relevant to Google.
5. Enter/Select the CA details.

Field description for CA details

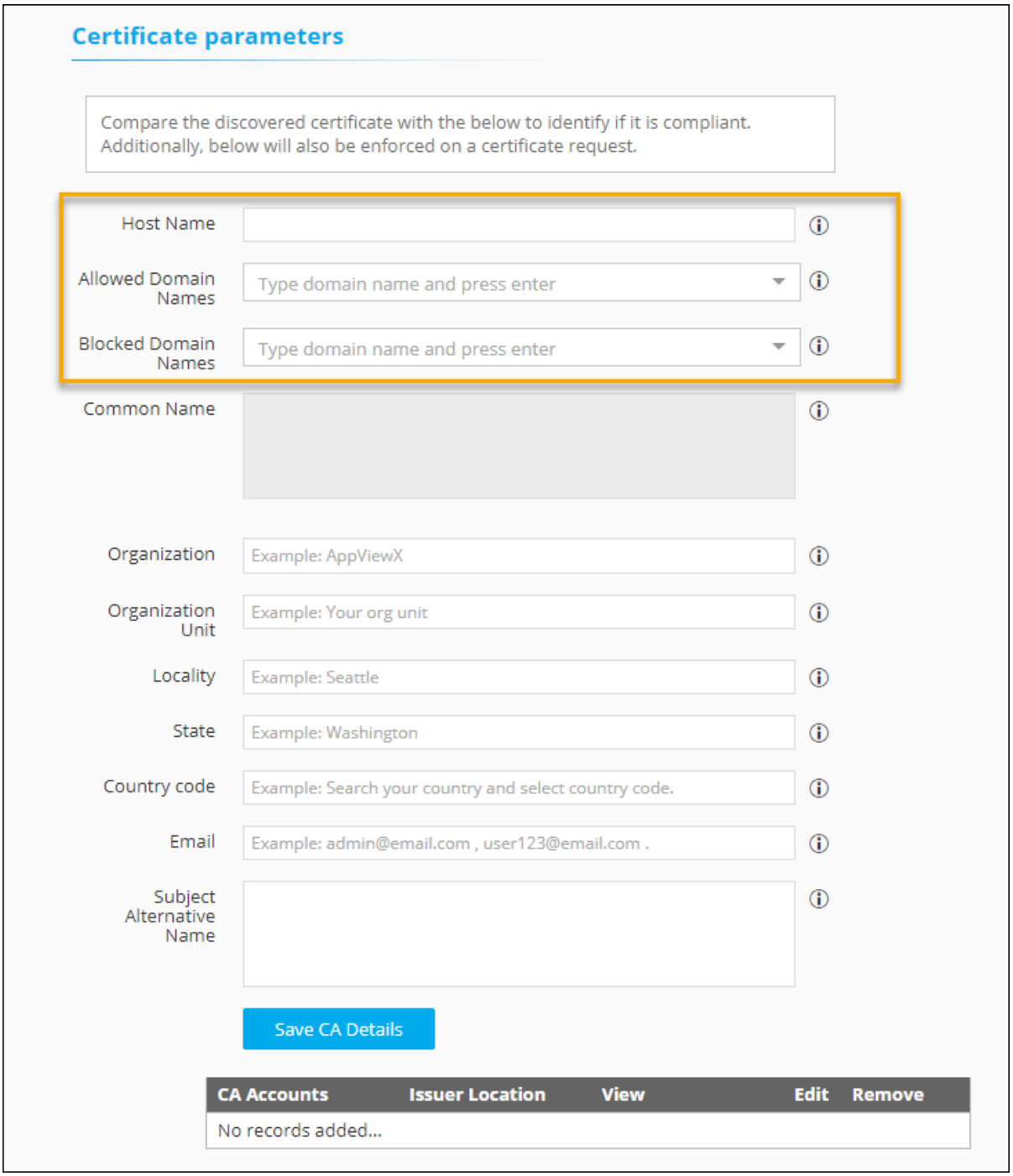
Field	Description
***CA Accounts**	The Google CA accounts configured in the CA settings screen are listed. Select a CA account from the list to create the policy.
***Issuer Location**	The issuer locations corresponding to the selected CA account are listed. Select an issuer location from the list to create the policy.
***Issuer Name**	The issuer names corresponding to the selected CA account are listed. Select an issuer name from the list to create the policy.
***Validity**	In the **Days**, **Month**, and **Year** dropdown lists, enter the validity period(s) for the certificate. You can enter more than one validity period in days/months/years, and one can then be chosen from the entered values at the time of certificate enrollment.
***Bit Length - Key Type**	From the dropdown list, select one (or more than one), bit length- key type pair(s). The discovered certificate's Key Type and Bit length will be compared against the selected B bit length- key type pair(s) to check for complaince with the policy. The

Field	Description
	Selected bit length- key type pair(s) is enforced while performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
***Hash Function**	From the dropdown list, select one (or more) hash functions. The discovered certificate's Key Hash Algorithm will be compared against the selected hash function to check for compliance with the policy. The selected hash function(s) is enforced while performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
*: *Mandatory fields*	

6. Enter/Select the certificate parameters.
   For the Policy Enforcement Type = **Strict**



For the Policy Enforcement Type = **Suggestive**



Field description for certificate Parameters

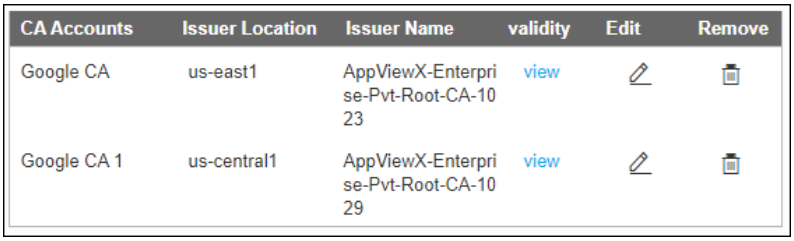
Field	Description
Restrict Wild Card Certificate	Slide toggle switch to the ON position to restrict the creation of wild card certificates using the policy.

Field	Description
Hostname	This text field is displayed if the **Policy Enforcement Type = Strict** or **Suggestive**. Enter the unique name or label for the host. The field is **mandatory** only when the **Policy Enforcement Type = Strict**. **Note:** The hostname should not start or end with a dot.
Allowed Domain Name	This text field is displayed if the **Policy Enforcement Type = Strict** or **Suggestive**. The field is **mandatory** only when the **Policy Enforcement Type = Strict**. Enter the valid domain name (two parts separated by a dot, such as example.com) .
Blocked Domain Name	This text field is displayed only if the **Policy Enforcement Type = Suggestive** Enter the domain names (two parts separated by a dot, such as example.com) that need to be blocked .
Common Name	Enter the common name. For example, ***.domain.com** This enforces domains for which a certificate can be requested. The common name is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**. **Note:** Use the * (asterisk) for the host part of the FQDN to enforce the domain. For example, ***.domain.com** will only allow users to request certificates with domain.com. Allowed special characters: ***(Asterisk), **-** (hyphen), **.** (period)

Field	Description
	.
Organization	You can provide the organization's name. The discovered certificate's **Subject Organization** will be compared against the organization provided in the policy to identify if they are complaints. **The organization** is enforced while performing any certificate request operations such as New, Renew, Regenerate.
Organization Unit	You can provide an organization unit. The discovered certificate's **Subject Organization Unit** will be compared against the organization unit provided in the policy to identify if they are Complaint. **Organization Unit** is enforced while performing any certificate request operations such as New, Renew, Regenerate.
Locality	You can provide a locality. The discovered certificate's **Locality** will be compared against the locality provided in the policy to identify if they are complaints. **The locality** is enforced while performing any certificate request operations such as New, Renew, Regenerate.
State	You can provide state. The discovered certificate's **State** will be compared against the state provided in the policy to identify if they are complaints. **The state** is enforced while performing any certificate request operations such as New, Renew, Regenerate.
Country code	You can provide a country code. The discovered certificate's **Country code** will be compared against the country code provided in the policy to identify if they are complaints. **Country code** is enforced while performing any certificate request operations such as New, Renew, Regenerate.

Field	Description
Email	You can provide an organization unit mail address. The discovered certificate's **mail address** will be compared against the email address provided in the policy to identify if they are Complaint. **Mail address** is enforced while performing any certificate request operations such as New, Renew, Regenerate.
Subject Alternative Name	You can provide the subject alternative name (SAN) It helps enforce additional domains for which a certificate can be requested. **Subject Alternative Name** is enforced while performing certificate request operations such as New, Renew, and Regenerate. **Note:** Use Asterisk (*) for the host part of the FQDN to enforce the domain. For example, *.domain.com will only allow users to request certificates with domain domain.com. Allowed Special Characters: Asterisk (*), Hyphen (-), Period (.), At (@)
*: *Mandatory fields*	

You can use the **Edit** option in the table to modify the configuration and **Remove** option to delete the configuration.



CA Accounts	Issuer Location	Issuer Name	validity	Edit	Remove
Google CA	us-east1	AppViewX-Enterprise-Pvt-Root-CA-1023	view		
Google CA 1	us-central1	AppViewX-Enterprise-Pvt-Root-CA-1029	view		

7. Click **Save CA Details** to save the configuration.
   A green tick mark is displayed in the **Certificate Authority** pane against **Google** to indicate the details are successfully stored.
8. From the **Group selection**, select one or more groups to map to the policy.
9. From the **Compliance Check** section, to perform an immediate compliance check, enable **Perform Compliance check**.

> **Note:** A scheduled compliance check will run periodically based on the settings defined in the job scheduler.

10. Click **Create Policy** button to create a new policy.
    The policy is created and a confirmation message is displayed.

Configuring Policy for HashiCorp Vault CA

Before You Begin

- Certificate Group(s) must be available to map the Policy to them.
- CA accounts (settings) must be available to which the policy is going to be created.
- Key Algorithm, Encryption Type must be available under the CA accounts.
- AppViewX permission required (Accounts > Roles - Click here to check Accounts management).

To configure a Hashicorp Vault CA policy:

1. Go to (**Menu**) > **SIGN+** > **GROUPS & POLICIES** > **CA Policy**.
   The **CA Policy** page is displayed with a list of policies and their associated groups.

> **Note:** A **Default** policy will always be present in the list. Most of the roles are mapped to this policy. This policy can be used for any of the configured CAs.

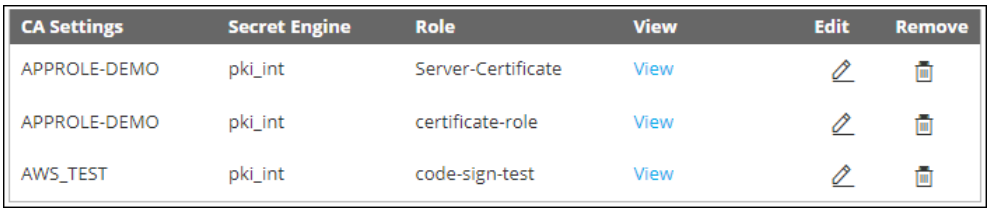
2. To create a custom policy, click **+Create** from the top-right corner of the **CA Policy** page.
   The **CA Policy:: Create** page is displayed.
3. Refer the Configuring Policy Details section in the SIGN+ Admin Guide to configure the following:
   - **Policy Details**
   - **Group Selection**
   - **Compliance Check**
4. In the **CA Details** section, from the **Certificate Authority** list in the left, select **HashicorpVault**.
   The **CA Details** section is updated to display fields relevant to HashicorpVault.
5. Enter/Select the CA details.

Field Description for CA Details

Field	Description
***CA Accounts**	Select a CA account from the list to create the policy.
***Secret Engine**	The single-select dropdown contains all the secret engines associated with the account. In a secret engine, a role describes an identity with a set of permissions, groups, or policies you want to attach to a user of the secret engine. User identity is often mapped to a specific role. Hence, a single secret engine needs to be selected to populate role (below) specific to it.
***Role**	The dropdown list contains all the roles mapped to the secret engine.
*: *Mandatory fields*	

6. Click **Add**.

   The CA details are saved to the table and the confirmation message is displayed.

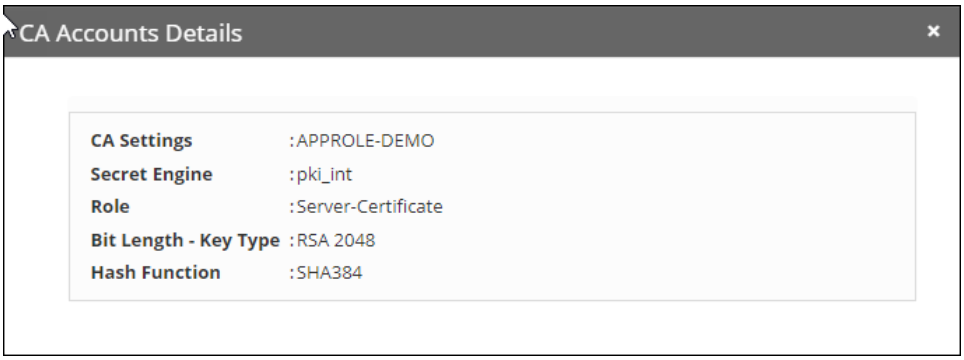


CA Settings	Secret Engine	Role	View	Edit	Remove
APPROLE-DEMO	pki_int	Server-Certificate	View		
APPROLE-DEMO	pki_int	certificate-role	View		
AWS_TEST	pki_int	code-sign-test	View		

   Multiple values can be configured based on the available CA settings and secret engines with different bit length - key type and hash function. The supported values include:

   - **Key Type**: RSA, EC
   - **Bit Length:**
     - *RSA key type*: 2048 (default), 3072, or 4096
     - *EC key type*: 224, 256 (default), 384, or 521
   - **Hash Function**: SHA-256, SHA-384, SHA-512

   The CA Details table has options to **View**, **Edit**, and **Delete**.



a. To view the CA details, click the View link in the View column - The CA account details are displayed in a pop-up window with the *Bit Length - Key Type* and *Hash Function.*
b. To update the CA details, select the edit icon in the Edit column.
c. To delete the CA details, select the delete icon.

7. Enter/Select the certificate parameters.

Field description for certificate parameters

Field	Description
Restrict Wild Card Certificate	Slide toggle switch to the ON position to restrict the creation of wild card certificates using the policy.
Host name	Enter the host name. The host name cannot start and end with a **.** (period)
***Allowed Domain Names**	Enter only the white-listed domain names. Press enter after adding the domain name. Multiple domain names can be added.
Common Name	Enter the common name. For example, ***.domain.com** This enforces domains for which a certificate can be requested. The common name is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**. **Note:** Use the ***** (asterisk) for the host part of the FQDN to enforce the domain. For example, ***.domain.com** will only allow users to request certificates with domain.com. Allowed special characters: ***(Asterisk), **-** (hyphen), **.** (period)

Field	Description
	.
Organization	Enter the organization name. The discovered certificate's subject organization will be compared against the organization provided in the policy to check for compliance. The organization is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Organization Unit	Enter the organization unit. The discovered certificate's Subject Organization Unit will be compared against the organization unit provided in the policy to check for compliance. Organization Unit is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Locality	Enter the locality name. The discovered certificate's locality will be compared against locality provided in the policy to check for compliance. The locality is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
State	Enter the state. The discovered certificate's state will be compared against the state provided in the policy to check for compliance. The state is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Country code	Enter the country code. The discovered certificate's country code will be compared against the country code provided in the policy to check for compliance. Country code is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Email	Enter the email address of the organization unit. The discovered certificate's email address will be compared against the email address provided in the policy to check for compliance. The email address is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.

Field	Description
Subject Alternative Name	Enter the subject alternative name (SAN). It helps enforce additional domains for which a certificate can be requested. The SAN is enforcedat the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**. **Note:** Use the ***** (asterisk) for the host part of the FQDN to enforce the domain. For example, ***.domain.com** will only allow users to request certificates with domain.com. Allowed special characters: *****(Asterisk), **-** (hyphen), **.** (period)
*: *Mandatory fields*	

8. Click **Save CA Details**.
   A green tick mark is displayed in the **Certificate Authority** pane against the **Hashicorp Vault** option to indicate that the details are successfully saved.
9. In the **Group Selection**, select one or more groups to map to the policy. Refer to the **Certificate Group** section to add/update groups.
10. Under the **Compliance Check** section, enable the **Perform Compliance Check** option to perform an immediate compliance check.
11. Click **Create Policy** button.
    The policy is created and a confirmation message is displayed.

Configuring Policy for HydrantID CA

1. Go to (**Menu**) > **SIGN+** > **GROUPS & POLICIES** > **CA Policy**.
   The **CA Policy** page is displayed.
2. Click **+ Create** from the top-right corner of the page.
   The **CA Policy:: Create** page is displayed.
3. Refer the Configuring Policy Details section in the SIGN+ Admin Guide to configure the following:
   - **Policy Details**
   - **Group Selection**
   - **Compliance Check**
4. In the **CA Details** section, from the **Certificate Authority** list in the left, select **HydrantID**.
   The **CA Details** section is updated to display fields relevant to HydrantID.
5. Enter/Select the CA details.

Field Description for CA Details

Field	Description
***CA Accounts**	The HydrantID CA accounts configured in the CA settings screen are listed here. Select a CA account from the list to create the policy.
***HydrantID Policy**	Policies associated with the account will be displayed here. Select a policy from the dropdown (single select).
***Validity**	In the **Days**, **Month**, and **Year** dropdown lists, enter the validity period(s) for the certificate. You can enter more than one validity period in days/months/years, and one can then be chosen from the entered values at the time of certificate enrollment.
*: *Mandatory fields*	

6. Click **Add**.
   The CA details are saved to the table and the confirmation message is displayed.

   You can use the **Edit** option in the table to modify the configuration and **Remove** option to delete the configuration.
7. From the ***Bit Length - Key Type** dropdown list, select one (or more than one), bit length- key type pair(s).
   The discovered certificate's Key Type and Bit length will be compared against the selected B bit length- key type pair(s) to check for complaince with the policy. The Selected bit length- key type pair(s) is enforced while performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
8. From the ***Hash Function** dropdown list, select one (or more) hash functions.
   The discovered certificate's Key Hash Algorithm will be compared against the selected hash function to check for compliance with the policy. The selected hash function(s) is enforced while performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
9. Enter/Select the **Certificate parameters** values.

Field description for certificate parameters

Field	Description
Restrict Wild Card Certificate	Slide toggle switch to the ON position to restrict the creation of wild card certificates using the policy.
Host name	Enter the host name. The host name cannot start and end with a **.** (period)
***Allowed Domain Names**	Enter only the white-listed domain names. Press enter after adding the domain name. Multiple domain names can be added.
Common Name	Enter the common name. For example, ***.domain.com** This enforces domains for which a certificate can be requested. The common name is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**. **Note:** Use the * (asterisk) for the host part of the FQDN to enforce the domain. For example, ***.domain.com** will only allow users to request certificates with domain.com. Allowed special characters: *(Asterisk), **-** (hyphen), **.** (period) .
Organization	Enter the organization name. The discovered certificate's subject organization will be compared against the organization provided in the policy to check for compliance. The organization is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Organization Unit	Enter the organization unit. The discovered certificate's Subject Organization Unit will be compared against the organization unit provided in the policy to check for compliance. Organization Unit is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Locality	Enter the locality name.

Field	Description
	The discovered certificate's locality will be compared against locality provided in the policy to check for compliance. The locality is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
State	Enter the state. The discovered certificate's state will be compared against the state provided in the policy to check for compliance. The state is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Country code	Enter the country code. The discovered certificate's country code will be compared against the country code provided in the policy to check for compliance. Country code is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Email	Enter the email address of the organization unit. The discovered certificate's email address will be compared against the email address provided in the policy to check for compliance. The email address is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Subject Alternative Name	Enter the subject alternative name (SAN). It helps enforce additional domains for which a certificate can be requested. The SAN is enforcedat the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**. **Note:** Use the ***** (asterisk) for the host part of the FQDN to enforce the domain. For example, ***.domain.com** will only allow users to request certificates with domain.com. Allowed special characters: *****(Asterisk), **-** (hyphen), **.** (period)
*: *Mandatory fields*	

10. Click **Save CA Details** button to save the configuration.
    A green tick mark will be displayed in the **Certificate Authority** pane against the **HydrantID** CA option to indicate that the details are successfully stored.

11. From the **Group selection**, select one or more groups to map to the policy.
12. From the **Compliance Check** section, to perform an immediate compliance check, enable **Perform Compliance check**.

> **Note:** A scheduled compliance check will run periodically based on the settings defined in the job scheduler.

13. Click **Create Policy** to create a new policy.
    The policy is created and a confirmation message is displayed.

Configuring Policy for Let's Encrypt CA

1. Go to (**Menu**) > **SIGN+** > **GROUPS & POLICIES** > **CA Policy**.
   The **CA Policy** page is displayed.
2. Click **+ Create** from the top-right corner of the page.
   The **CA Policy: Create** page is displayed.
3. Refer the Configuring Policy Details section in the SIGN+ Admin Guide to configure the following:
   - **Policy Details**
   - **Group Selection**
   - **Compliance Check**
4. In the **CA Details** section, from the **Certificate Authority** list in the left, select **LetsEncrypt**.
   The **CA Details** section is updated to display fields relevant to LetsEncrypt.
5. Enter/Select the CA details.

Field Description for CA Details

Field	Description
***CA Accounts**	The Let's Encrypt CA accounts configured in CA settings are listed here. Select a CA account from the list to create the policy.
***Validity**	In the **Days**, **Month**, and **Year** dropdown lists, enter the validity period(s) for the certificate. You can enter more than one validity period in days/months/years, and one can then be chosen from the entered values at the time of certificate enrollment.
*: *Mandatory fields*	

6. Click **Add**.
   The CA details are saved to the table and the confirmation message is displayed.

   You can use the **Edit** option in the table to modify the configuration and **Remove** option to delete the configuration.
7. From the ***Bit Length - Key Type** dropdown list, select one (or more than one), bit length- key type pair(s).
   The discovered certificate's Key Type and Bit length will be compared against the selected B bit length- key type pair(s) to check for complaince with the policy. The Selected bit length- key type pair(s) is enforced while performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
8. From the ***Hash Function** dropdown list, select one (or more) hash functions.
   The discovered certificate's Key Hash Algorithm will be compared against the selected hash function to check for compliance with the policy. The selected hash function(s) is enforced while performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
9. Enter/Select the **Certificate parameters** values.

Field description for certificate parameters

Field	Description
Restrict Wild Card Certificate	Slide toggle switch to the ON position to restrict the creation of wild card certificates using the policy.
Host name	Enter the host name. The host name cannot start and end with a **.** (period)
***Allowed Domain Names**	Enter only the white-listed domain names. Press enter after adding the domain name. Multiple domain names can be added.
Common Name	Enter the common name. For example, ***.domain.com** This enforces domains for which a certificate can be requested. The common name is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.

Field	Description
	Note: Use the ***** (asterisk) for the host part of the FQDN to enforce the domain. For example, ***.domain.com** will only allow users to request certificates with domain.com. Allowed special characters: *****(Asterisk), **-** (hyphen), **.** (period) .
Organization	Enter the organization name. The discovered certificate's subject organization will be compared against the organization provided in the policy to check for compliance. The organization is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Organization Unit	Enter the organization unit. The discovered certificate's Subject Organization Unit will be compared against the organization unit provided in the policy to check for compliance. Organization Unit is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Locality	Enter the locality name. The discovered certificate's locality will be compared against locality provided in the policy to check for compliance. The locality is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
State	Enter the state. The discovered certificate's state will be compared against the state provided in the policy to check for compliance. The state is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Country code	Enter the country code. The discovered certificate's country code will be compared against the country code provided in the policy to check for compliance. Country code is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Email	Enter the email address of the organization unit.

Field	Description
	The discovered certificate's email address will be compared against the email address provided in the policy to check for compliance. The email address is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Subject Alternative Name	Enter the subject alternative name (SAN). It helps enforce additional domains for which a certificate can be requested. The SAN is enforcedat the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**. **Note:** Use the ***** (asterisk) for the host part of the FQDN to enforce the domain. For example, ***.domain.com** will only allow users to request certificates with domain.com. Allowed special characters: *****(Asterisk), **-** (hyphen), **.** (period)
*: *Mandatory fields*	

10. Click **Save CA Details** button to save the configuration.
    A green tick mark will be displayed in the **Certificate Authority** pane against **LetsEncrypt** to indicate that the details are successfully stored.

11. From the **Group selection**, select one or more groups to map to the policy.
12. From the **Compliance Check** section, to perform an immediate compliance check, enable **Perform Compliance check**.

Note: A scheduled compliance check will run periodically based on the settings defined in the job scheduler.

13. Click **Create Policy** to create a new policy.
    The policy is created and a confirmation message is displayed.

Configuring Policy for Microsoft Enterprise CA

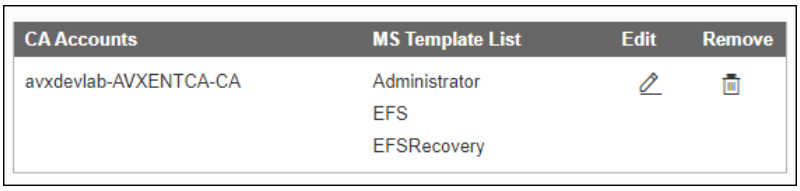
1. Go to (**Menu**) > **SIGN+** > **GROUPS & POLICIES** > **CA Policy**.
   The **CA Policy** page is displayed.
2. Click **+ Create** from the top-right corner of the page.
   The **CA Policy: Create** page is displayed.
3. Refer the Configuring Policy Details section in the SIGN+ Admin Guide to configure the following:
   - **Policy Details**
   - **Group Selection**
   - **Compliance Check**
4. In the **CA Details** section, from the **Certificate Authority** list in the left, select **Microsoft Enterprise**.
   The **CA Details** section is updated to display fields relevant to Microsoft Enterprise.
5. Enter/Select the CA details.

Field Description for CA Details

Field	Description
***CA Accounts**	The Microsoft Enterprise CA accounts configured in the CA settings are listed. Select a CA account from the list to create the policy.
***MS Template List**	The MS templates configured for the selected CA account are listed. Select MS template(s) from the list to associate with the policy.
*: *Mandatory fields*	

6. Click **Add**.
   The CA details are saved to the table and the confirmation message is displayed.

   You can use **the Edit** option in the table to modify the configuration and **Remove** option to delete the configuration.



7. From the ***Bit Length - Key Type** dropdown list, select one (or more than one), bit length- key type pair(s).

The discovered certificate's Key Type and Bit length will be compared against the selected B bit length- key type pair(s) to check for complaince with the policy. The Selected bit length- key type pair(s) is enforced while performing any certificate request operations such as **New**, **Renew**, **Regenerate**.

8. From the ***Hash Function** dropdown list, select one (or more) hash functions.
   The discovered certificate's Key Hash Algorithm will be compared against the selected hash function to check for compliance with the policy. The selected hash function(s) is enforced while performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
9. Enter/Select the **Certificate parameters** values.

Field description for certificate parameters

Field	Description
Restrict Wild Card Certificate	Slide toggle switch to the ON position to restrict the creation of wild card certificates using the policy.
Host name	Enter the host name. The host name cannot start and end with a **.** (period)
***Allowed Domain Names**	Enter only the white-listed domain names. Press enter after adding the domain name. Multiple domain names can be added.
Common Name	Enter the common name. For example, ***.domain.com** This enforces domains for which a certificate can be requested. The common name is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**. **Note:** Use the * (asterisk) for the host part of the FQDN to enforce the domain. For example, ***.domain.com** will only allow users to request certificates with domain.com. Allowed special characters: *(Asterisk), **-** (hyphen), **.** (period) .
Organization	Enter the organization name.

Field	Description
	The discovered certificate's subject organization will be compared against the organization provided in the policy to check for compliance. The organization is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Organization Unit	Enter the organization unit. The discovered certificate's Subject Organization Unit will be compared against the organization unit provided in the policy to check for compliance. Organization Unit is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Locality	Enter the locality name. The discovered certificate's locality will be compared against locality provided in the policy to check for compliance. The locality is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
State	Enter the state. The discovered certificate's state will be compared against the state provided in the policy to check for compliance. The state is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Country code	Enter the country code. The discovered certificate's country code will be compared against the country code provided in the policy to check for compliance. Country code is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Email	Enter the email address of the organization unit. The discovered certificate's email address will be compared against the email address provided in the policy to check for compliance. The email address is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Subject Alternative Name	Enter the subject alternative name (SAN). It helps enforce additional domains for which a certificate can be requested. The SAN is enforcedat the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.

Field	Description
	Note: Use the * (asterisk) for the host part of the FQDN to enforce the domain. For example, ***.domain.com** will only allow users to request certificates with domain.com. Allowed special characters: *(Asterisk), - (hyphen), . (period)
*: *Mandatory fields*	

10. Click **Save CA Details** button to save the configuration.
    A green tick mark will be displayed in the **Certificate Authority** pane against **Microsoft Enterprise** to indicate the details are successfully stored.

11. From the **Group selection**, select one or more groups to map to the policy.
12. From the **Compliance Check** section, to perform an immediate compliance check, enable **Perform Compliance check**.

> **Note:** A scheduled compliance check will run periodically based on the settings defined in the job scheduler.

13. Click **Create Policy** to create a new policy.
    The policy is created and a confirmation message is displayed.

Configuring Policy for Microsoft Standalone CA

1. Go to (**Menu**) > **SIGN+** > **GROUPS & POLICIES** > **CA Policy**.
   The **CA Policy** page is displayed.
2. Click **+ Create** from the top-right corner of the page.
   The **CA Policy: Create** page is displayed.
3. Refer the Configuring Policy Details section in the SIGN+ Admin Guide to configure the following:
   - **Policy Details**
   - **Group Selection**
   - **Compliance Check**
4. In the **CA Details** section, from the **Certificate Authority** list in the left, select **Microsoft Standalone**.

The **CA Details** section is updated to display fields relevant to Microsoft Standalone.

5. Enter/Select the CA details.

Field description for CA Details

Field	Description
***CA Accounts**	The Microsoft Standalone CA accounts configured in the CA settings are listed. Select a CA account from the list to create the policy.
*: *Mandatory fields*	

6. Click **Add**.
   The CA details are saved to the table and the confirmation message is displayed.

   You can use **the Edit** option in the table to modify the configuration and **Remove** option to delete the configuration.

7. From the ***Bit Length - Key Type** dropdown list, select one (or more than one), bit length- key type pair(s).
   The discovered certificate's Key Type and Bit length will be compared against the selected B bit length- key type pair(s) to check for complaince with the policy. The Selected bit length- key type pair(s) is enforced while performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
8. From the ***Hash Function** dropdown list, select one (or more) hash functions.
   The discovered certificate's Key Hash Algorithm will be compared against the selected hash function to check for compliance with the policy. The selected hash function(s) is enforced while performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
9. Enter/Select the **Certificate parameters** values.

Field description for certificate parameters

Field	Description
Restrict Wild Card Certificate	Slide toggle switch to the ON position to restrict the creation of wild card certificates using the policy.
Host name	Enter the host name. The host name cannot start and end with a **.** (period)

Field	Description
***Allowed Domain Names**	Enter only the white-listed domain names. Press enter after adding the domain name. Multiple domain names can be added.
Common Name	Enter the common name. For example, ***.domain.com** This enforces domains for which a certificate can be requested. The common name is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**. **Note:** Use the * (asterisk) for the host part of the FQDN to enforce the domain. For example, ***.domain.com** will only allow users to request certificates with domain.com. Allowed special characters: *(Asterisk), **-** (hyphen), **.** (period) .
Organization	Enter the organization name. The discovered certificate's subject organization will be compared against the organization provided in the policy to check for compliance. The organization is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Organization Unit	Enter the organization unit. The discovered certificate's Subject Organization Unit will be compared against the organization unit provided in the policy to check for compliance. Organization Unit is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Locality	Enter the locality name. The discovered certificate's locality will be compared against locality provided in the policy to check for compliance. The locality is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
State	Enter the state.

Field	Description
	The discovered certificate's state will be compared against the state provided in the policy to check for compliance. The state is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Country code	Enter the country code. The discovered certificate's country code will be compared against the country code provided in the policy to check for compliance. Country code is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Email	Enter the email address of the organization unit. The discovered certificate's email address will be compared against the email address provided in the policy to check for compliance. The email address is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Subject Alternative Name	Enter the subject alternative name (SAN). It helps enforce additional domains for which a certificate can be requested. The SAN is enforcedat the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**. **Note:** Use the ***** (asterisk) for the host part of the FQDN to enforce the domain. For example, ***.domain.com** will only allow users to request certificates with domain.com. Allowed special characters: *****(Asterisk), **-** (hyphen), **.** (period)
*: *Mandatory fields*	

10. Click **Save CA Details** button to save the configuration.
    A green tick mark will be displayed in the **Certificate Authority** pane against **Microsoft Standalone** to indicate the details are successfully stored.
11. From the **Group selection**, select one or more groups to map to the policy.
12. From the **Compliance Check** section, to perform an immediate compliance check, enable **Perform Compliance check**.

Note: A scheduled compliance check will run periodically based on the settings defined in the job scheduler.

13. Click **Create Policy** to create a new policy.
    The policy is created and a confirmation message is displayed.

Configuring Policy for Nexus CA

1. Go to (**Menu**) > **SIGN+** > **GROUPS & POLICIES** > **CA Policy**.
   The **CA Policy** page is displayed.
2. Click **+ Create** from the top-right corner of the page.
   The **CA Policy:: Create** page is displayed.
3. Refer the Configuring Policy Details section in the SIGN+ Admin Guide to configure the following:
   - **Policy Details**
   - **Group Selection**
   - **Compliance Check**
4. In the **CA Details** section, from the **Certificate Authority** list in the left, select **Nexus**.
   The **CA Details** section is updated to display fields relevant to Nexus.
5. Enter/Select the CA details.

Field Description for CA Details

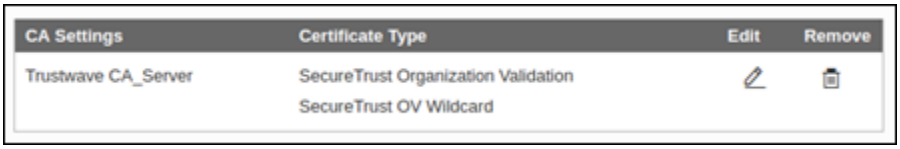
Field	Description
***CA Accounts**	The Nexus CA accounts configured in the CA settings screen are listed. Select a CA account from the list to create the policy.
***Validity**	In the **Days**, **Month**, and **Year** dropdown lists, enter the validity period(s) for the certificate. You can enter more than one validity period in days/ months/years, and one can then be chosen from the entered values at the time of certificate enrollment.
*: *Mandatory fields*	

6. Once the CA account and the validity fields are populated, a new section called **Vendor Specific Details** is enabled. Select the **Procedure** field from the drop-down list and click **Add** to save the CA details to the table. You can also use the **Remove** option to delete the configuration.
   Select the details as follows:
   a. **Case 1 -** Select **Server** check box, the procedures listed in the **Procedures** dropdown will be - Mapped to servers and default procedures.
   b. **Case 2 -** Select **Client** check box, the procedures listed in the **Procedures** dropdown will be - mapped to client and default procedures.
   c. **Case 3 -** Select **Server** and **Client** check box, the procedures listed in the **Procedures** dropdown will be - mapped to server, client, and default procedures.

   Note: The procedures displayed in the dropdown should have the **cert type** appended in the value. For example: - *Procedure name_Server/client/default/Code-Signing*.

7. Click **Add**. The CA details are saved to the table and the confirmation message displays.
   The CA details are saved to the table and the confirmation message is displayed.

   You can use **the Edit** option in the table to modify the configuration and **Remove** option to delete the configuration.



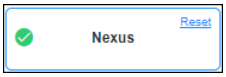
8. From the ***Bit Length - Key Type** dropdown list, select one (or more than one), bit length- key type pair(s).
   The discovered certificate's Key Type and Bit length will be compared against the selected B bit length- key type pair(s) to check for complaince with the policy. The Selected bit length- key type pair(s) is enforced while performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
9. From the ***Hash Function** dropdown list, select one (or more) hash functions.
   The discovered certificate's Key Hash Algorithm will be compared against the selected hash function to check for compliance with the policy. The selected hash function(s) is enforced while performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
10. Enter/Select the **Certificate parameters** values.

Field description for certificate parameters

Field	Description
Host name	Enter the host name. The host name cannot start and end with a **.** (period)
***Allowed Domain Names**	Enter only the white-listed domain names. Press enter after adding the domain name. Multiple domain names can be added.
Common Name	Enter the common name. For example, ***.domain.com** This enforces domains for which a certificate can be requested. The common name is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**. **Note:** Use the ***** (asterisk) for the host part of the FQDN to enforce the domain. For example, ***.domain.com** will only allow users to request certificates with domain.com. Allowed special characters: ***(Asterisk), **-** (hyphen), **.** (period) .
Organization	Enter the organization name. The discovered certificate's subject organization will be compared against the organization provided in the policy to check for compliance. The organization is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Organization Unit	Enter the organization unit. The discovered certificate's Subject Organization Unit will be compared against the organization unit provided in the policy to check for compliance. Organization Unit is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Locality	Enter the locality name. The discovered certificate's locality will be compared against locality provided in the policy to check for compliance. The locality is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.

Field	Description
State	Enter the state. The discovered certificate's state will be compared against the state provided in the policy to check for compliance. The state is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Country code	Enter the country code. The discovered certificate's country code will be compared against the country code provided in the policy to check for compliance. Country code is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Email	Enter the email address of the organization unit. The discovered certificate's email address will be compared against the email address provided in the policy to check for compliance. The email address is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Subject Alternative Name	Enter the subject alternative name (SAN). It helps enforce additional domains for which a certificate can be requested. The SAN is enforcedat the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**. **Note:** Use the ***** (asterisk) for the host part of the FQDN to enforce the domain. For example, ***.domain.com** will only allow users to request certificates with domain.com. Allowed special characters: *****(Asterisk), **-** (hyphen), **.** (period)
*: *Mandatory fields*	

11. Click **Save CA Details** button to save the configuration.
    A green tick mark will be displayed in the **Certificate Authority** pane against **Nexus** to indicate that the details are successfully stored.



12. From the **Group selection**, select one or more groups to map to the policy.

13. From the **Compliance Check** section, to perform an immediate compliance check, enable **Perform Compliance check**.

    Note: A scheduled compliance check will run periodically based on the settings defined in the job scheduler.

14. Click **Create Policy** to create a new policy.

    The policy is created and a confirmation message is displayed.

Configuring Policy for OpenTrust CA

1. Go to (**Menu**) > **SIGN+** > **GROUPS & POLICIES** > **CA Policy**.

   The **CA Policy** page is displayed.
2. Click **+ Create** from the top-right corner of the page.

   The **CA Policy: Create** page is displayed.
3. Refer the Configuring Policy Details section in the SIGN+ Admin Guide to configure the following:
   - **Policy Details**
   - **Group Selection**
   - **Compliance Check**
4. In the **CA Details** section, from the **Certificate Authority** list in the left, select **OpenTrust**.

   The **CA Details** section is updated to display fields relevant to OpenTrust.
5. Enter/Select the CA details.

   Field Description for CA Details

Field	Description
***CA Account**	The OpenTrust CA accounts configured in the CA settings screen are listed. Select a CA account from the list to create the policy.
***Certificate Management Profile**	Select the certificate management profile from the dropdown list.
***Zone**	Select the zone from the dropdown list.
*: *Mandatory fields*	

6. Enter/Select the **Profile Parameters**.

Field Description for profile parameters

Field	Description
***Common Name**	Enter a common name for the policy.
Organizational Unit	Enter the organizational unit.
Organization	Enter the name of the organization.
*: *Mandatory fields*	

7. Click **Add** button.
   The CA details are saved to the table and the confirmation message is displayed.

   You can use the **Remove** option to delete the configuration.
8. From the ***Bit Length - Key Type** dropdown list, select one (or more than one), bit length- key type pair(s).
   The discovered certificate's Key Type and Bit length will be compared against the selected B bit length- key type pair(s) to check for complaince with the policy. The Selected bit length- key type pair(s) is enforced while performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
9. From the ***Hash Function** dropdown list, select one (or more) hash functions.
   The discovered certificate's Key Hash Algorithm will be compared against the selected hash function to check for compliance with the policy. The selected hash function(s) is enforced while performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
10. Enter/Select the **Certificate parameters** values.

Field description for certificate parameters

Field	Description
Restrict Wild Card Certificate	Slide toggle switch to the ON position to restrict the creation of wild card certificates using the policy.
Host name	Enter the host name. The host name cannot start and end with a **.** (period)
***Allowed Domain Names**	Enter only the white-listed domain names. Press enter after adding the domain name. Multiple domain names can be added.

Field	Description
Common Name	Enter the common name. For example, ***.domain.com** This enforces domains for which a certificate can be requested. The common name is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**. **Note:** Use the * (asterisk) for the host part of the FQDN to enforce the domain. For example, ***.domain.com** will only allow users to request certificates with domain.com. Allowed special characters: *(Asterisk), **-** (hyphen), **.** (period) .
Organization	Enter the organization name. The discovered certificate's subject organization will be compared against the organization provided in the policy to check for compliance. The organization is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Organization Unit	Enter the organization unit. The discovered certificate's Subject Organization Unit will be compared against the organization unit provided in the policy to check for compliance. Organization Unit is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Locality	Enter the locality name. The discovered certificate's locality will be compared against locality provided in the policy to check for compliance. The locality is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
State	Enter the state. The discovered certificate's state will be compared against the state provided in the policy to check for compliance. The state is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Country code	Enter the country code.

Field	Description
	The discovered certificate's country code will be compared against the country code provided in the policy to check for compliance. Country code is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Email	Enter the email address of the organization unit. The discovered certificate's email address will be compared against the email address provided in the policy to check for compliance. The email address is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Subject Alternative Name	Enter the subject alternative name (SAN). It helps enforce additional domains for which a certificate can be requested. The SAN is enforcedat the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**. **Note:** Use the ***** (asterisk) for the host part of the FQDN to enforce the domain. For example, ***.domain.com** will only allow users to request certificates with domain.com. Allowed special characters: *****(Asterisk), **-** (hyphen), **.** (period)
**: Mandatory fields*	

11. Click **Save CA Details** button to save the configuration.
    A green tick mark displays in the **Certificate Authority** pane against **OpenTrust** to indicate that the details are successfully stored.
12. From the **Group selection**, select one or more groups to map to the policy.
13. From the **Compliance Check** section, to perform an immediate compliance check, enable **Perform Compliance check**.

    Note: A scheduled compliance check will run periodically based on the settings defined in the job scheduler.

14. Click **Create Policy** to create a new policy.
    The policy is created and a confirmation message is displayed.

Configuring Policy for Sectigo CA

1. Go to (**Menu**) > **SIGN+** > **GROUPS & POLICIES** > **CA Policy**.
   The **CA Policy** page is displayed.
2. Click **+ Create** from the top-right corner of the page.
   The **CA Policy:: Create** page is displayed.
3. Refer the Configuring Policy Details section in the SIGN+ Admin Guide to configure the following:
   - **Policy Details**
   - **Group Selection**
   - **Compliance Check**
4. In the **CA Details** section, from the **Certificate Authority** list in the left, select **Comodo Certificate Manager**.
   The **CA Details** section is updated to display fields relevant to Comodo Certificate Manager.
5. Enter/Select the CA details.

Field description for CA Details

Field	Description
***CA Accounts**	The Sectigo CA accounts configured in the CA settings screen are listed. Select a CA account from the list to create the policy.
***Certificate Type**	The certificate types corresponding to the selected CA account are listed. Select one (or) more certificate type(s) from the list to create the policy.
***Validity**	In the **Days**, **Month**, and **Year** dropdown lists, enter the validity period(s) for the certificate. You can enter more than one validity period in days/months/years, and one can then be chosen from the entered values at the time of certificate enrollment.
*: *Mandatory fields*	

6. Click **Add**.
   The CA details are saved to the table and the confirmation message is displayed.

   You can use **the Edit** option in the table to modify the configuration and **Remove** option to delete the configuration.

7. From the ***Bit Length - Key Type** dropdown list, select one (or more than one), bit length- key type pair(s).
   The discovered certificate's Key Type and Bit length will be compared against the selected B bit length- key type pair(s) to check for complaince with the policy. The Selected bit length- key type pair(s) is enforced while performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
8. From the ***Hash Function** dropdown list, select one (or more) hash functions.
   The discovered certificate's Key Hash Algorithm will be compared against the selected hash function to check for compliance with the policy. The selected hash function(s) is enforced while performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
9. Enter/Select the **Certificate parameters** values.

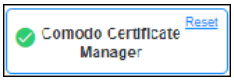
Field description for certificate parameters

Field	Description
Restrict Wild Card Certificate	Slide toggle switch to the ON position to restrict the creation of wild card certificates using the policy.
Host name	Enter the host name. The host name cannot start and end with a **.** (period)
***Allowed Domain Names**	Enter only the white-listed domain names. Press enter after adding the domain name. Multiple domain names can be added.
Common Name	Enter the common name. For example, ***.domain.com** This enforces domains for which a certificate can be requested. The common name is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.

Field	Description
	Note: Use the ***** (asterisk) for the host part of the FQDN to enforce the domain. For example, ***.domain.com** will only allow users to request certificates with domain.com. Allowed special characters: *****(Asterisk), **-** (hyphen), **.** (period) .
Organization	Enter the organization name. The discovered certificate's subject organization will be compared against the organization provided in the policy to check for compliance. The organization is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Organization Unit	Enter the organization unit. The discovered certificate's Subject Organization Unit will be compared against the organization unit provided in the policy to check for compliance. Organization Unit is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Locality	Enter the locality name. The discovered certificate's locality will be compared against locality provided in the policy to check for compliance. The locality is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
State	Enter the state. The discovered certificate's state will be compared against the state provided in the policy to check for compliance. The state is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Country code	Enter the country code. The discovered certificate's country code will be compared against the country code provided in the policy to check for compliance. Country code is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Email	Enter the email address of the organization unit.

Field	Description
	The discovered certificate's email address will be compared against the email address provided in the policy to check for compliance. The email address is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Subject Alternative Name	Enter the subject alternative name (SAN). It helps enforce additional domains for which a certificate can be requested. The SAN is enforcedat the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**. **Note:** Use the ***** (asterisk) for the host part of the FQDN to enforce the domain. For example, ***.domain.com** will only allow users to request certificates with domain.com. Allowed special characters: *****(Asterisk), **-** (hyphen), **.** (period)
*: *Mandatory fields*	

10. Click **Save CA Details** button to save the configuration.
    A green tick mark is displayed in the **Certificate Authority** pane against **Comodo Certificate Manager** to indicate that the details are successfully stored.



11. From the **Group selection**, select one or more groups to map to the policy.
12. From the **Compliance Check** section, to perform an immediate compliance check, enable **Perform Compliance check**.

Note: A scheduled compliance check will run periodically based on the settings defined in the job scheduler.

13. Click **Create Policy** to create a new policy.
    The policy is created and a confirmation message is displayed.

Configuring Policy for Symantec CA

1. Go to (**Menu**) > **SIGN+** > **GROUPS & POLICIES** > **CA Policy**.
   The **CA Policy** page is displayed.
2. Click **+ Create** from the top-right corner of the page.
   The **CA Policy:: Create** page is displayed.
3. Refer the Configuring Policy Details section in the SIGN+ Admin Guide to configure the following:
   - **Policy Details**
   - **Group Selection**
   - **Compliance Check**
4. In the **CA Details** section, from the **Certificate Authority** list in the left, select **Symantec**.
   The **CA Details** section is updated to display fields relevant to Symantec.
5. Enter/Select the CA details.

Field description for CA Details

Field	Description
***CA Account**	The Symantec CA accounts configured in CA settings screen are listed. Select a CA account from the list to create the policy.
***Certificate Type**	The certificate types corresponding to the selected CA account are listed. Select one (or) more certificate type(s) from the list to create the policy.
***Validity**	In the **Days**, **Month**, and **Year** dropdown lists, enter the validity period(s) for the certificate. You can enter more than one validity period in days/months/years, and one can then be chosen from the entered values at the time of certificate enrollment.
*: *Mandatory fields*	

6. In the **Vendor Specific Details section**, select the server type from the dropdown list.
7. Click **Add**.
   The CA details are saved to the table and the confirmation message is displayed.

   You can use **the Edit** option in the table to modify the configuration and **Remove** option to delete the configuration.
8. From the ***Bit Length - Key Type** dropdown list, select one (or more than one), bit length- key type pair(s).

The discovered certificate's Key Type and Bit length will be compared against the selected B bit length- key type pair(s) to check for complaince with the policy. The Selected bit length- key type pair(s) is enforced while performing any certificate request operations such as **New**, **Renew**, **Regenerate**.

9. From the ***Hash Function** dropdown list, select one (or more) hash functions.
   The discovered certificate's Key Hash Algorithm will be compared against the selected hash function to check for compliance with the policy. The selected hash function(s) is enforced while performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
10. Enter/Select the **Certificate parameters** values.

Field description for certificate parameters

Field	Description
Restrict Wild Card Certificate	Slide toggle switch to the ON position to restrict the creation of wild card certificates using the policy.
Host name	Enter the host name. The host name cannot start and end with a **.** (period)
***Allowed Domain Names**	Enter only the white-listed domain names. Press enter after adding the domain name. Multiple domain names can be added.
Common Name	Enter the common name. For example, ***.domain.com** This enforces domains for which a certificate can be requested. The common name is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**. **Note:** Use the * (asterisk) for the host part of the FQDN to enforce the domain. For example, ***.domain.com** will only allow users to request certificates with domain.com. Allowed special characters: *(Asterisk), **-** (hyphen), **.** (period) .
Organization	Enter the organization name.

Field	Description
	The discovered certificate's subject organization will be compared against the organization provided in the policy to check for compliance. The organization is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Organization Unit	Enter the organization unit. The discovered certificate's Subject Organization Unit will be compared against the organization unit provided in the policy to check for compliance. Organization Unit is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Locality	Enter the locality name. The discovered certificate's locality will be compared against locality provided in the policy to check for compliance. The locality is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
State	Enter the state. The discovered certificate's state will be compared against the state provided in the policy to check for compliance. The state is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Country code	Enter the country code. The discovered certificate's country code will be compared against the country code provided in the policy to check for compliance. Country code is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Email	Enter the email address of the organization unit. The discovered certificate's email address will be compared against the email address provided in the policy to check for compliance. The email address is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Subject Alternative Name	Enter the subject alternative name (SAN). It helps enforce additional domains for which a certificate can be requested. The SAN is enforcedat the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.

Field	Description
	Note: Use the * (asterisk) for the host part of the FQDN to enforce the domain. For example, ***.domain.com** will only allow users to request certificates with domain.com. Allowed special characters: *(Asterisk), - (hyphen), . (period)
*: *Mandatory fields*	

11. Click **Save CA Details** button to save the configuration.
    A green tick mark is displayed in the **Certificate Authority** pane against **Symantec** to indicate that the details are successfully stored.

12. From the **Group selection**, select one or more groups to map to the policy.
13. From the **Compliance Check** section, to perform an immediate compliance check, enable **Perform Compliance check**.

> **Note:** A scheduled compliance check will run periodically based on the settings defined in the job scheduler.

14. Click **Create Policy** button to create a new policy.
    The policy is created and a confirmation message is displayed.

Configuring Policy for Trustwave CA

1. Go to (**Menu**) > **SIGN+** > **GROUPS & POLICIES** > **CA Policy**.
   The **CA Policy** page is displayed.
2. Click **+ Create** from the top-right corner of the page.
   The **CA Policy:: Create** page is displayed.
3. Refer the Configuring Policy Details section in the SIGN+ Admin Guide to configure the following:
   - **Policy Details**
   - **Group Selection**
   - **Compliance Check**
4. In the **CA Details** section, from the **Certificate Authority** list in the left, select **Trustwave**.

The **CA Details** section is updated to display fields relevant to Trustwave.

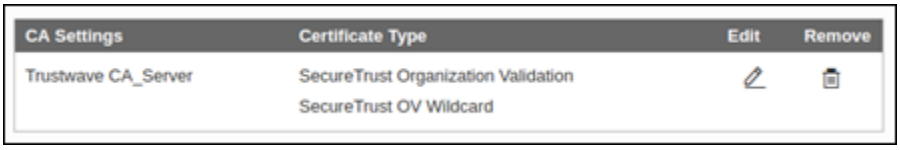
5. Enter/Select the CA details.

Field description for CA Details

Field	Description
***CA Account**	The Trustwave CA accounts configured in CA settings screen are listed. Select a CA account from the list to create the policy.
***Certificate Type**	The certificate types corresponding to the selected CA account are listed. Select one (or) more certificate type(s) from the list to create the policy.
***Validity**	In the **Days**, **Month**, and **Year** dropdown lists, enter the validity period(s) for the certificate. You can enter more than one validity period in days/months/years, and one can then be chosen from the entered values at the time of certificate enrollment.
*: *Mandatory fields*	

6. Click **Add**.
   The CA details are saved to the table and the confirmation message is displayed.

   You can use **the Edit** option in the table to modify the configuration and **Remove** option to delete the configuration.



7. From the ***Bit Length - Key Type** dropdown list, select one (or more than one), bit length- key type pair(s).
   The discovered certificate's Key Type and Bit length will be compared against the selected B bit length- key type pair(s) to check for complaince with the policy. The Selected bit length- key type pair(s) is enforced while performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
8. From the ***Hash Function** dropdown list, select one (or more) hash functions.
   The discovered certificate's Key Hash Algorithm will be compared against the selected hash function to check for compliance with the policy. The selected hash function(s) is enforced while performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
9. Enter/Select the **Certificate parameters** values.

Field description for certificate parameters

Field	Description
Restrict Wild Card Certificate	Slide toggle switch to the ON position to restrict the creation of wild card certificates using the policy.
Host name	Enter the host name. The host name cannot start and end with a **.** (period)
***Allowed Domain Names**	Enter only the white-listed domain names. Press enter after adding the domain name. Multiple domain names can be added.
Common Name	Enter the common name. For example, ***.domain.com** This enforces domains for which a certificate can be requested. The common name is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**. **Note:** Use the * (asterisk) for the host part of the FQDN to enforce the domain. For example, ***.domain.com** will only allow users to request certificates with domain.com. Allowed special characters: *(Asterisk), **-** (hyphen), **.** (period) .
Organization	Enter the organization name. The discovered certificate's subject organization will be compared against the organization provided in the policy to check for compliance. The organization is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Organization Unit	Enter the organization unit. The discovered certificate's Subject Organization Unit will be compared against the organization unit provided in the policy to check for compliance. Organization Unit is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Locality	Enter the locality name.

Field	Description
	The discovered certificate's locality will be compared against locality provided in the policy to check for compliance. The locality is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
State	Enter the state. The discovered certificate's state will be compared against the state provided in the policy to check for compliance. The state is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Country code	Enter the country code. The discovered certificate's country code will be compared against the country code provided in the policy to check for compliance. Country code is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Email	Enter the email address of the organization unit. The discovered certificate's email address will be compared against the email address provided in the policy to check for compliance. The email address is enforced at the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**.
Subject Alternative Name	Enter the subject alternative name (SAN). It helps enforce additional domains for which a certificate can be requested. The SAN is enforcedat the time of performing any certificate request operations such as **New**, **Renew**, **Regenerate**. **Note:** Use the ***** (asterisk) for the host part of the FQDN to enforce the domain. For example, ***.domain.com** will only allow users to request certificates with domain.com. Allowed special characters: *****(Asterisk), **-** (hyphen), **.** (period)
*: *Mandatory fields*	

10. Click **Save CA Details** button to save the configuration.
    A green tick mark is displayed in the **Certificate Authority** pane against **Trustwave** to indicate that the details are successfully stored.

11. From the **Group selection**, select one or more groups to map to the policy.
12. From the **Compliance Check** section, to perform an immediate compliance check, enable **Perform Compliance check**.

> **Note:** A scheduled compliance check will run periodically based on the settings defined in the job scheduler.

13. Click **Create Policy** button to create a new policy.
    The policy is created and a confirmation message is displayed.

Signing Policy

A signing policy, in the context of code signing and security practices, refers to a set of rules, guidelines, and procedures that govern how digital signatures are applied to software, scripts, or other digital assets.

Signing policies are typically defined and implemented within organizations to ensure the secure and consistent application of digital signatures. These policies are a fundamental part of a broader security strategy and are important for various reasons:

Security Assurance: Signing policies help ensure the security of software and digital assets by specifying who can sign code, what can be signed, and under what circumstances. They establish a framework for mitigating risks associated with unauthorized or malicious code modifications.

Authentication: Signing policies often dictate the use of code signing certificates issued by trusted certificate authorities (CAs). These certificates verify the identity of the signer, adding a layer of authentication to the signed code. This helps establish trust in the source of the software.

Integrity: Policies define the conditions under which code should be signed. By adhering to these policies, organizations maintain the integrity of their codebase, as any unauthorized changes or tampering will result in the invalidation of the digital signature.

Non-Repudiation: Code signing with adherence to policies provides non-repudiation, meaning that the signer cannot deny their involvement in the signing process. This is crucial for accountability and legal purposes.

Compliance: Many industries and regulatory bodies require organizations to adhere to specific code signing practices. Signing policies helps ensure compliance with these regulations, which is especially important in sectors like healthcare, finance, and government.

Version Control: Policies can specify how versioning should be managed for signed code. This helps users verify the authenticity and integrity of software updates and patches.

Key Aspects Covered by Signing Policies

A signing policy plays a crucial role in an organization's cybersecurity strategy by fostering trust, preserving code integrity, and mitigating the risk of malware and security breaches. It provides explicit guidelines for secure code signing practices, making it an essential component of secure software development and distribution. Key aspects of security addressed by signing policies include:

Authorized Signers: Signing policies are used to determine authorized personnel, identifying individuals within the organization authorized to sign code or digital assets, which may include specific developers or security team members

Signing Environment: Secure code signing environments identify and include environments and systems that are secure and trusted.

Certificate Usage: Managing code signing certificates addresses the selection and management of thecertificates, often emphasizing the use of certificates issued by recognized Certificate Authorities (CAs).

Review and Approval: Code review and approval procedures ensure compliance with security and quality standards before signing.

Timestamping: Signing policies ensure valid signatures over time, implementing timestamping requirements to maintain the validity of signatures, even after the certificate's expiration.

Revocation: Signing policies outline procedures for revoking signatures in cases of compromised certificates or unauthorized code changes.

• Configuring Signing Policy

Configuring Signing Policy

1. Go to **(Menu) > SIGN+ > GROUPS & POLICIES > Signing Policy**.
   The **Signing Policy** page is displayed.
2. From the top-right corner of the page, click **Create**.

3. Enter/select the **Policy details**.

Field description for the Policy Details section

Field name	Description
***Policy Name**	Provide a unique name for the signing policy. No special characters other than **'.', '-','_'** are allowed. The name should not start with special characters.
***Hash Function**	Select the hash function you want to configure for code signing: [Dropdown Options - **SHA-1**, **SHA-256**, **SHA-384**, **SHA-512**]
Timestamping	Choose a trusted timestamping authority from the dropdown list: [Dropdown Options - **DigiCert, Entrust, Global Sign, IdenTrust, Sectigo, Other, None**]. If you choose **Other**, kindly provide the **timestamping URL**. **Note:** If you select **None**, the **Timestamping** will not be applied to the configured signing policy.
***Signing Type**	Choose between **Hash Based** or **File Based** signing
***File Types**	This field is displayed only when the **Signing Type** is set as **File Based**. Select one or more file types that should be signed using the signing policy. Supported file types include **PS1, EXE, CAT, MSI, JS, JAR, APK, VBS, CAB, WSF, DLL, PSM1, PSD1, PS1XML, JSE, and VBE** among others. **Note:** Selected file types will only be permitted for upload and signing under this policy. **Note:** Signing operations for the HSM-based certificates for the script files will be supported by upgrading the JSign Version from 3.0 to 6.0.

Field name	Description
	Restriction: CAT files do not work with HSM-based certificates, but works for a File Based certificates.
Data Center	This field is displayed only when the **Signing Type** is set as **File Based**. Select the data center from the dropdown where timestamping requests will be routed. Ensure that the selected TSA URL is reachable from all servers managed under the chosen data center.
Restriction Type	Select **None** or between **IP-based restriction** or **IP range-based restriction**.
***List of IP's**	This field is displayed when the **Restriction Type** is set as **IP**. If you selected **IP-based restriction**, enter a list of valid individual IP addresses at subnet or system level.
***Start IP** ***End IP**	This field is displayed when the **Restriction Type** is set as **IP Range**. If you selected an **IP range-based restriction**, enter the start and end IP addresses, ensuring the end IP is greater than the start IP.
Enable HSM Polling	This applies to HSM-based certificates. Enable the toggle to allow the system to retry fetching the signing operation status based on the configurations defined in the signing policy, overriding the global Sign Settings. **Note: Enable HSM Polling** option is enabled by default for all existing signing policies.
***Number Of Polls**	This field is displayed when the **Enable HSM Polling** toggle is enabled. Add the number of polls if the certificate is based on HSM, and Specify the total number of polls to be conducted within the designated polling interval and the value must be an integer between 1 and 20.

Field name	Description
***Polling Interval**	This field is displayed when the **Enable HSM Polling** toggle is enabled. Add the Polling Interval if the certificate is based on HSM, Set the time interval between consecutive polls and the value must be an integer between 1 and 300000 milliseconds.
Test Policy	Select the checkbox to create the policy for internal testing. Enabling this option ignores all signatures associated with the policy in the license counting.
Enable Email notification	Enable the toggle button to receive email notifications and updates via email when the signing events occur.
*: *Mandatory fields*	

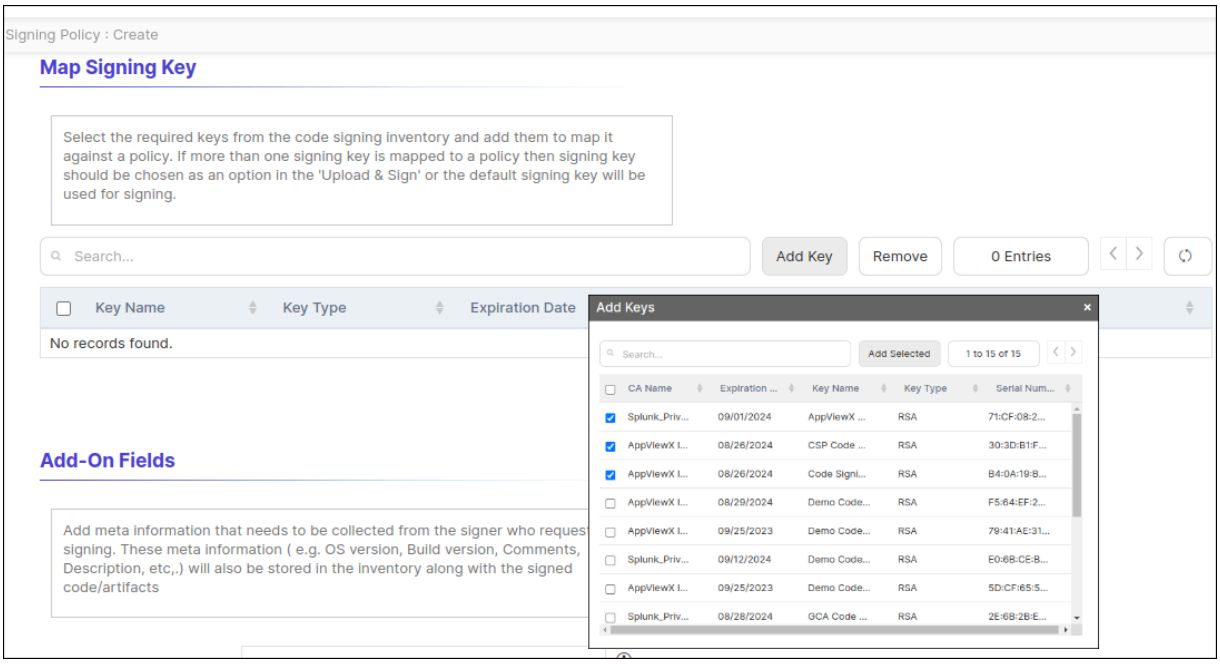
4. *(Optional step)* If the **Enable Email notification** toggle switch is enabled then enter/select the **Email Configuration** details as described below.

Field description for the Email Configuration section

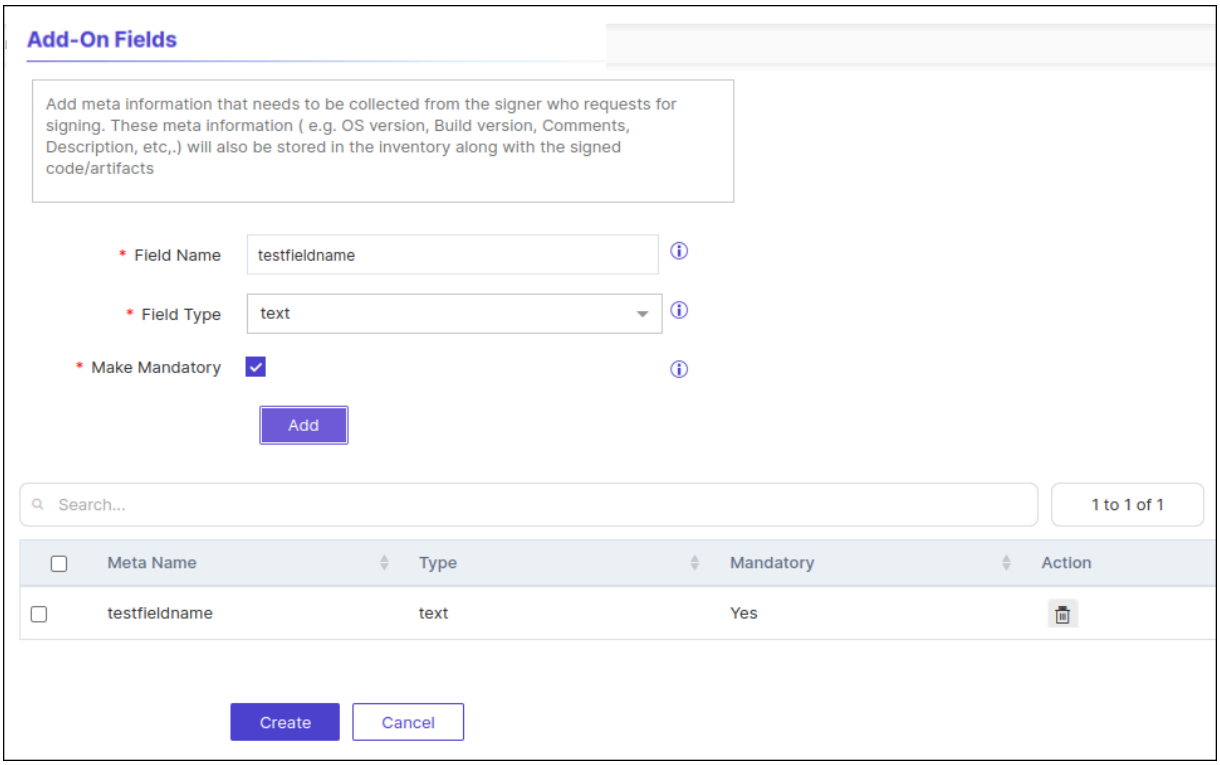
Field name	Description
***Email Subject**	Enter the subject line for the email notification to identify the purpose or content of the email. Acceptable characters are letters, numbers, and spaces.
***To**	Enter one or more recipients email address separated by comma.
Event Type	Choose the type of events for which notifications are required. The values are **Success**, **Failure**, or **Both**.
***Required Field**	A multi-select dropdown field with values - **Policy name**, **Signing Type**, **Key Name**, **IP Address**, **Signing Time**, and **Username**. Select one or more values whose details are to be displayed in the mail body for comprehensive notification.
*: *Mandatory fields*	

5. In the **Map Signing Key** section, select the required keys from the code signing inventory and add them to map them against a policy as shown in the below images. If more than one signing key is

mapped to a policy then the signing key should be chosen as an option in the Upload & Sign or the default signing key will be used for signing. Click the **Add Key** button to add the keys.



6. In the **Add-On Fields** section, add meta information that needs to be collected from the signer who requests for signing. This meta information (e.g. OS version, build version, comments, description, etc,.) will also be stored in the inventory along with the signed code/artifacts. Enter values in the **Field Name** and **Field Type** fields and select the **Make Mandatory** checkbox as required.



7. Click **Add**.

   The **Add-On Fields** will be added in the meta information table.
8. Click **Create**.

   The signing policy is created in the inventory.

> **Note:** Deletion of a signing policy is restricted if it is associated with a signing record.

What to do next:

- Upload and sign the code signing file with the specified file type selected during policy creation.

Sign Logs

In the realm of code signing, audit logs play a fundamental role in ensuring the security, integrity, and transparency of the entire signing process. These logs

- Serve as a comprehensive record of critical activities and events related to code signing operations
- Provide invaluable insights into the provenance and legitimacy of code signatures
- The instrumental in helping organizations meet compliance requirements, prevent security breaches, and trace the history of code signing activities.

Audit logs within the context of code signing encompass a wide array of information, including the

- Details of code signature creation, updates, and other relevant information
- Essential data such as the digital certificates used in the signing process
- Timestamps of each operation
- The identity of the signatory.

By preserving this information, audit logs enable organizations to verify the authenticity of signed code and to identify any suspicious or unauthorized activities that may compromise the security of their software. The visibility provided by these logs is crucial for both internal security practices and compliance with regulatory standards, ultimately ensuring the trustworthiness of code and the safety of end-users.

Viewing Sign Logs

1. Go to (**Menu**) > **SIGN+** > **Sign Logs** > **Sign Logs**.
   The **Logging** page is displayed with the **Sign** tab open by default.
2. Use the following filters to display limited data:
   - Search by text field
   - Search by time icon (date and time)
   - Search by **Method of Login** dropdown.

   This page displays the following details:

Field description of Sign Logs table

Fields	Description
Time	Date and time at which the activity was carried out.
User	Username of the user who performed the activity.
Severity	Severity of the activity logged (Notification, Debug, Warn, Error, Fatal, Critical).

Fields	Description
Category	Name of the module. In this case - Sign.
Method of login	Indicates the type of signing. UI is for a file-based signing (File upload and sign) and API is for a hash-based sign.
Log message	Description of the activity logged.
Source IP	IP of the device from where the action was performed.
AppViewX node	IP:node of the AppViewX server from where the action was performed.

Exporting Logs

AppViewX lets you export logs as Excel sheets.

1. Go to (**Menu**) > **SIGN+** > **Sign Logs** > **Sign Logs**.
   The **Logging** page is displayed with the **Sign** tab open by default.
2. From the top right corner of the page, click **(Export)** icon.
3. Navigate to the location to save the log file, and click **Save**.
   All logs of the sign type are downloaded and saved.

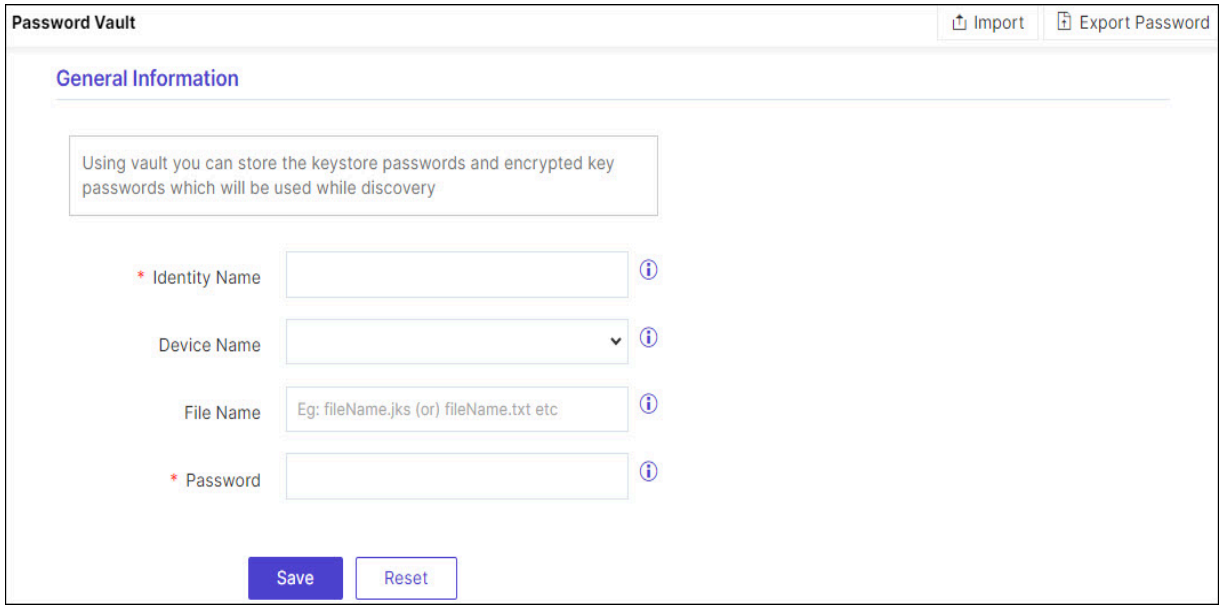
Password Vault

The password vault is used to store all certificate passwords of selected ADC devices. All the password-protected certificates that are discovered, will be decrypted and pushed to the discovery grid in the AppViewX Inventory. This happens only if passwords are matched with passwords that are stored in the vault.

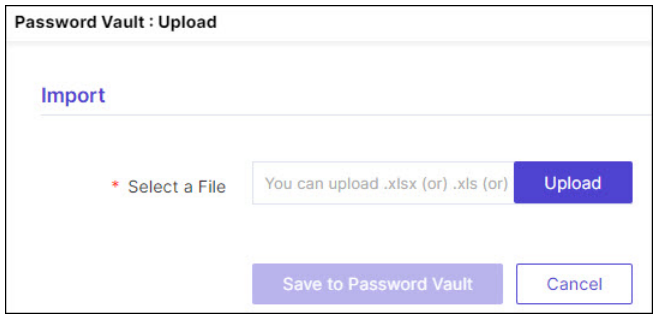
Before you Begin: To decrypt the password-protected certificates, ensure that you have a valid certificate password.

1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Password Vault**.
   The **Password Vault** page is displayed.
2. In the **General Information** section, enter an **Identity Name** for the password you want to add in the vault.

3. From the **Device Name** dropdown, select the ADC device whose password-protected certificate details you want to store.
4. In the **File Name** field, enter a certificate file name to help users identify it.
5. In the **Password** field, enter the password that is associated with the certificate.
6. Click **Save**.
7. To import a file (in XLS or CSV format) with a list of all certificate passwords, from the top-right corner of the page, click **Import**.



The **Password Vault : Upload** page is displayed.



This option is used to store the certificate passwords directly in the vault instead of adding them manually.

8. To export all stored certificate passwords from the vault as a zip file to your local system, from the top-right corner of the page, click **Export Password**.
9. To modify the existing details, click **Edit.**
   To update the password, click **Update.**

   To delete the password details, click **Delete.**

Configuring Certificate Attributes and Tags

Adding Attribute Information

SIGN+ uses certificate attributes for creating additional placeholder fields that can be used to track a certificate. An administrator can create one or more fields that a requester enrolling a certificate can fill and use for future tracking.

Remember:

- Certificate attributes are CA or organization-specific attributes, apart from the CSR parameters.
- Once configured, these attributes will be shown to collect values during certificate enrollment.
- Business units specific parameters can be stored for quick filtering and auditing.

1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Attributes and Tags**.
   The **Certificate Attributes** page is displayed.
2. Click **Add New** from certificate attributes section.
   The **Certificate Attributes** pop-up window is displayed.
3. Enter/Select the **Certificate Attributes** values.

Field description for the certificate attributes configuration parameters

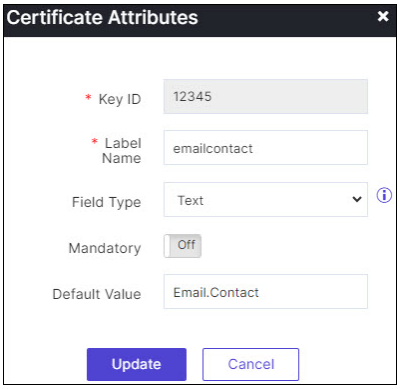
Field	Description
***Key ID**	Unique key for the attribute.
***Label Name**	Attribute name which will be shown during certificate enrollment. Eg. email contact, owner.
Field Type	Selected field type will be as text in the certificate Attribute type in the enrolment page.

Field	Description
Mandatory	Enable this field if the default value must be mandatory.
Default Value	Set a default value for the attribute.
*: *Mandatory fields*	

4. Click **Save**.
   The certificate attribute is added.

Updating Certificate Attributes

1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Attributes and Tags**.
   The **Certificate Attributes** page is displayed.
2. From the **Actions** column, click **(Edit)** icon.
   The **Certificate Attributes** update pop-up is displayed.
3. Edit the certificate attribute configuration.



4. Click **Update**.
   The certificate attribute is updated.

Deleting Certificate Attributes

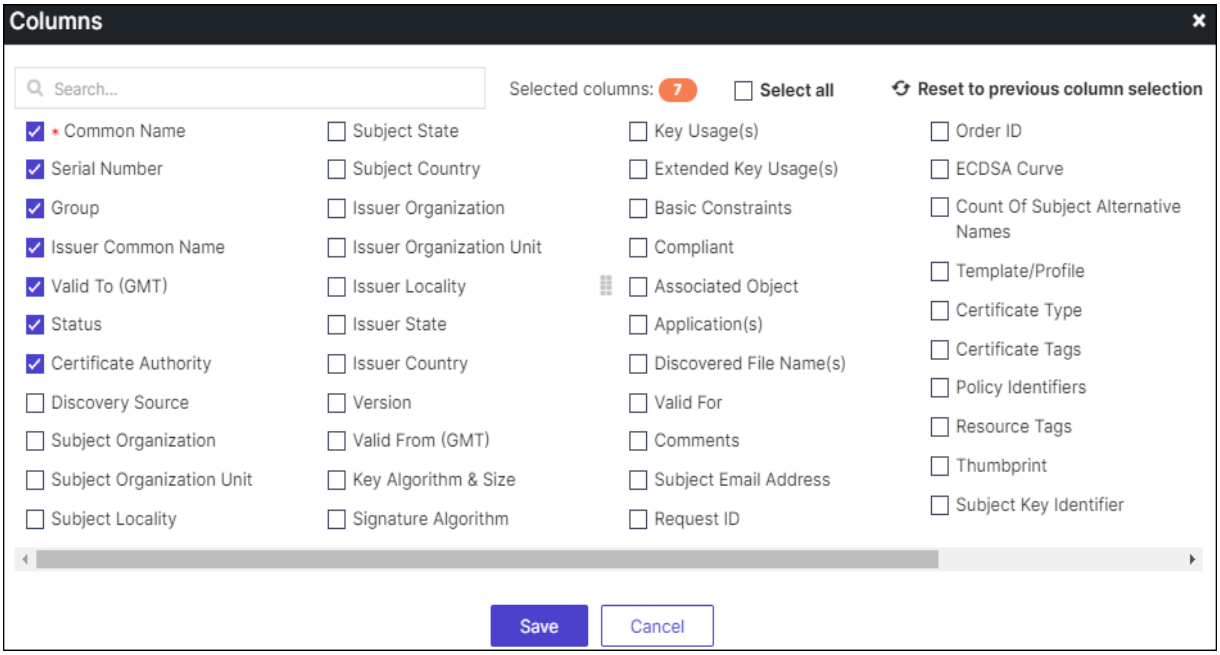
1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Certificate Attributes**.
   The **Certificate Attributes** page is displayed.
2. From the **Actions** column, click **(Delete)** icon.
   A confirmation dialog box is displayed.

3. Click **Delete**.
   The certificate attribute is deleted.

Viewing Certificate Attributes in Certificate Inventory

The configured attributes and their default values, if specified, will be shown in the certificate inventory along with the rest of the certificate details.

1. Go to **(Menu)** > **SIGN+** > **CERTIFICATE INVENTORY** > **Code Signing**.
   The **Code Signing Certificate** page is displayed.
2. Click Columns **(Columns)** icon.
   Select certificate attributes to display in the certificate inventory.



3. Click **Save**.
   The selected certificate attributes will be displayed in the certificate inventory.

Configuring Certificate Profiles

AppViewX **SIGN+** offers administrators the capability to define the type or purpose of a certificate through certificate profiles. An administrator can configure multiple profiles defining the key usage and extended key usage for a certificate enrolled through AppViewX. The profiles defined are applicable on certificates enrolled through AppViewX CA or Custom CA.

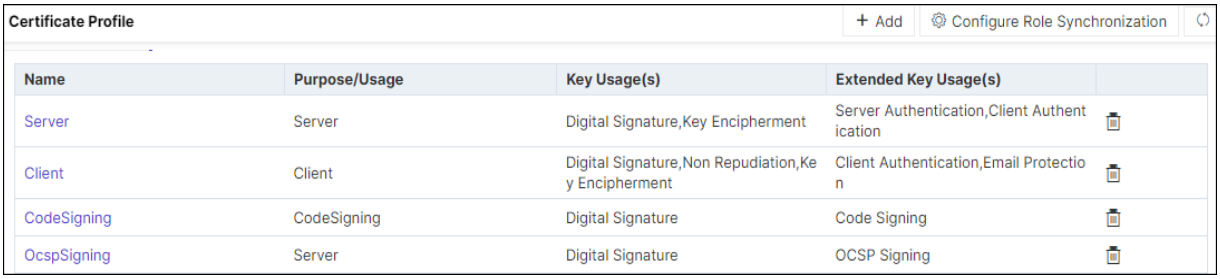
Note: An administrator can white label AppViewX CA through Custom CA.

Remember:

- Certificate profiles configure key usage extensions that define the purpose of the public key contained in a certificate.
- Once configured, these profiles will be used to define key usage and extended key usage while the signing a CSR through AppViewX CA and white labeled AppViewX CA or Custom CA.
- Sign+ comes prebuilt with profiles corresponding to a standard code signing certificate.

Adding a Certificate Profile

1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Certificate Profile**.
   The **Certificate Profile** page is displayed.



2. From the top-right corner of the screen, click **+ Add**.
3. Enter/Select the **General Information** details.

Field Description for General Information

Fields	Description
Name	Unique name to identify the profile.

Fields	Description
	Validation: Profile name should not start with special characters. Can contain only alphanumeric characters, no special characters except -, _, . are allowed.
Purpose/Usage	Certificate type to which the Key Usage extensions are signed with.

4. Configure **Key Usages** for the certificate profile.

Field Description for Key Usages

Fields	Description
Critical	Enable this field to sign the key usage extensions as **critical**.
***Key Usage(s)**	Key usage extensions with which the CSR is signed.
*: *Mandatory fields*	

5. Configure **Extended Key Usages** for the certificate profile.

Field Description for Extended Key Usages

Fields	Description
Critical	Enable this field to sign the extended key usage extensions as **critical**.
***Extended Key Usage(s)**	Extended key usage extensions with which the CSR is signed.
*: *Mandatory fields*	

6. Enter the **Policy ID**.
7. Click **Save**.
   The certificate profile is added.

Updating a Certificate Profile

To update certificate profile settings:

1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Certificate Profile**.
   The **Certificate Profile** page is displayed.

2. Click the **Name** of the profile to be edited.
3. Edit the certificate profile as required.
4. Click **Update**.
   The certificate profile is updated.

Deleting a Certificate Profile

To delete the certificate profile settings:

1. Go to (**Menu**) > **SIGN+** > **ADMINISTRATION** > **Certificate Profile**.
   The **Certificate Profile** page is displayed.
2. For the profile to be deleted, click the (**Delete**) icon.

   The **Delete Confirmation** popup is displayed.
3. Click **Yes**.
   The certificate profile is deleted.

Default User Roles and Permissions

SIGN+ offers a set of predefined roles designed to simplify user management, enforce security policies, and ensure that each user has access only to the necessary resources and functions required for their role. These roles are critical in maintaining the integrity of the system, ensuring proper access controls, and facilitating smooth operations across different user types.

The following roles are created by default within SIGN+:

1. **SIGN_Managers**
   - **Purpose:** The SIGN_Managers role is designed for users with managerial responsibilities within SIGN+. These users oversee signing policies, manage user roles, and have access to audit logs to monitor and review the platform's usage.
   - **Permissions:**

- Full access to all signing-related resources.
- Ability to create and modify signing policies.
- Manage and assign roles to other users.
- Access to audit trails, usage metrics, and reports.

2. **SIGN_Dev_Build_Users**
   - **Purpose:** SIGN_Dev_Build_Users role is meant for users who require signing capabilities as part of the development and build processes. These users primarily work with code and build artifacts that need to be signed to ensure integrity and authenticity.
   - **Permissions:**
     - Access to development and build signing functionalities.
     - Ability to manage personal certificates for code signing.
     - View-only access to signing policies and configuration settings.
3. **SIGN_Individual_Users**
   - **Purpose:** SIGN_Individual_Users role is for end users who need basic signing capabilities. These users typically sign documents or assets in the system but do not have any administrative or development responsibilities.
   - **Permissions:**
     - Basic signing capabilities for documents and files.
     - Access limited to signing assets assigned to them or their projects.
     - No access to user management, policy editing, or audit logs.
4. **SIGN_API_Users**
   - **Purpose:** SIGN_API_Users role is intended for users or applications that interact with SIGN + programmatically through APIs. This role is essential for automation or integration with other systems, enabling users to integrate SIGN+ capabilities into their own applications.
   - **Permissions:**
     - Access to SIGN+ API endpoints based on predefined API permissions.
     - Limited signing capabilities through API for designated assets.
     - No interactive access to the SIGN+ UI.

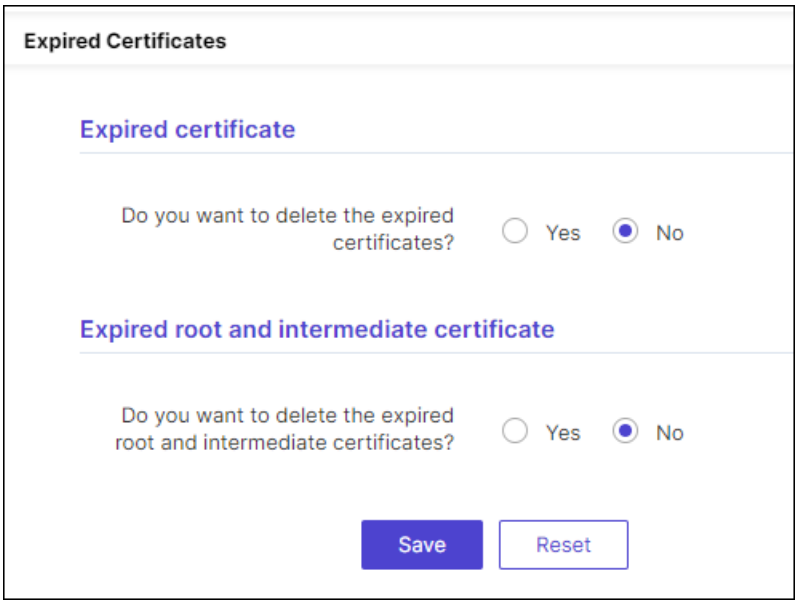
   Each of these roles comes with specific permissions tailored to their intended use cases, ensuring that users can perform their responsibilities effectively.

Expired Certificates

AppViewX SIGN+ gives you options to delete the expired certificates, along with the root and intermediate ones. Certificates are deleted after they are expired based on the configured date.

Tip: A best practice during certificate discovery is to apply a rule to avoid the discovery of expired certificates.

1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Expired Certificates**.
   The **Expired Certificates** page is displayed.



2. In the **Expired certificate** section, enter/select the details to configure the deletion of expired certificates.

Field	Description
***Number of days after expiry**	Enter the number of days after expiry after which the expired certificates will be deleted.
Backup Required	By default, this is set to **NO**. To enable the backup, select **Yes**. The fields **Backup Limit** and **Deletion Batch Limit** are displayed.
***Backup Limit**	Enter a numeric value for the backup limit of the expired certificates.
***Deletion Batch Limit**	Enter a numeric value for the deletion batch limit of the expired certificates.

Field	Description
Do you want to delete the certificates from all groups?	By default, the deletion of expired certificates from all group is enabled. To disable this, select **No**. The **Certificate Group** dropdown list is displayed.
***Certificate Group**	To delete the expired certificates from only specific groups,select the required certificate group(s) from this dropdown list.
*: *Mandatory fields*	

3. In the **Expired root and intermediate certificate** section, to delete the expired root and intermediate certificates, select **Yes**.
4. In the **Number of days after expiry** field, enter the number of days after certificate expiry after which the expired root and intermediate certificates will be deleted.

History of Certificates

SIGN+ lets you retain the history of old certificates before the renew/ regenerate/ reissue action. The history will be available in the inventory and can be tracked in the holistic view as well.

1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **History of Certificates**.
   The **History of Certificates** page is displayed.

   By default, the application retains the history of certificates.
2. To disable the retention of certificate history, click **No**.
3. In the **Number of days to delete certificates after its renew/reissue/regeneration** field, enter the number of days after which the certificates will be deleted.
4. Click **Save**.

Job Scheduler

The job scheduler lets you configure a host of tasks to mange certificate actions, perform validation checks, and monitor the logs for various actions.

List of Schedulable Tasks

S. No	Task Name	Description
1	Auto Regenerate Certificates	Automatically regenerates the configured certificates
2	Auto Renew Certificates	Automatically renewes the configured certificates
3	CA Connector Validity Updater	Updates the CA connector with maximum validity offered by the CA for server certificates
4	CRL Certificate Revocation Check	CRLs for all certificates in the inventory are downloaded for a revocation check
5	CRL Download Monitor Job	Monitors every five minutes to check for CRLs to download
6	Certificate Authority Connection Check	Checks connection with all configured Certificate Authority settings
7	Certificate CAA Record Check	Fetch data for certificate CAA report.
8	Certificate Expiry Status Check	Updates the expiry status in certificate every day
9	Certificate Polling Request	Configure the polling request for fetching certificate details from CA.
10	Certificate Revoke Status Check From CA	Retrieving the certificate revocation status issued or renewed by the CA.
11	Certificate Transparency Check	Fetch data for Certificate Transparency Report.
12	Certificate Vulnerability Check	Fetch the vulnerability details from certificate endpoints.
13	Certificate compliance check	Validate certificates in the inventory for compliance.
14	Certificate validation check	Validate trust details for all certificates in the inventory.
15	Deletes Expired Certificates	Deletes expired certificates
16	Delete Renew/Reissue/Regenerate Certificates	Deletes renewed/reissued/regenerated certificates

List of Schedulable Tasks (continued)

S. No	Task Name	Description
17	Device and Cert Sync Status job	Periodically executed to verify the status of the device and certificate sync job
18	Device and Certificate synchronization	Check synchronization of devices and their certificates.
19	Fetch End Entity Profile From CA	Retrieve end entity profiles from certificate authority.
20	Periodic CRL Update for AppViewX and Custom CAs	Updates CRL of AppViewX and Custom CAs revoked certificates.
21	Pkiaas AEP Purge log Job	Purge the user/device logs older than 24 hours.
22	Pkiaas OCSP Sync Job	Updates revoked cert details in cache
23	Update Certificate Cipher Suite Report	Updates the Certificate Cipher Suite Report
24	Update Certificate Expiration Report	Updates the Certificate Expiration Report
25	Update Certificate Orphan Report	Updates the Certificate Orphan Report
26	Update Count By Issuer Report	Updates the Count By Issuer Report
27	Update Expiry Report By Month	Updates the Expiry Report By Month
28	Update Policy Compliance Report	Updates the Policy Compliance Report
29	Update Report By Certificate Authority	Updates the Report By Certificate Authority
30	Update Report By Source	Updates the Report By Source
31	Update Stale Certificate Report	Updates the Stale Certificate Report
32	Update Validation status Report	Updates the Validation Status Report
33	Vulnerability CAA Record Generation Complete Scan	Vulnerability CAA Record Generation
34	Vulnerability CAA Record Generation Delta Scan	Vulnerability CAA Record Generation
35	Vulnerability Compliance Generation	Vulnerability Compliance Generation for ROCA and Key Strength

List of Schedulable Tasks (continued)

S. No	Task Name	Description
36	Vulnerability Endpoint Validation Generation	Vulnerability Endpoint Validation Generation - NMAP SCAN Heart Bleed, Poodle, Cipher TLS
37	Vulnerability Group Metrics Generation	Certificate Group Metrics Generation

The process to create a scheduled task is the same for all the above tasks. As an example, we'll schedule the **Auto Regenerate Certificates** task.

1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Job Scheduler**.
   The **Job Scheduler** page is displayed.
2. From the **Task Name** column, click the task you want to schedule. For this example, select **Auto Regenerate Certificates**.
   The **Auto Regenerate Certificates** pop-up window is displayed.
3. Enter/Select the details required to automatically regenerate certificates.

   Field description for parameters for automatically regenerating certificates

Field	Description
***Description**	This field is usually pre-populated with a text description. However, you can modify the description as required.
***Time Zone**	Select the required timezone that suits your requirement from the dropdown.
***Occurrence Type**	To define how frequently certificates will be auto regenerted, from the dropdown list, select a occurrence type .
***Starts on**	Based on the occurrence type selected, to schedule the task, use the calendar widget to set the date and time details.
Repeat	Select the number of times for which the task will be repeated, from the start date.
*: *Mandatory fields*	

4. Click **Update**.
   The task is updated with the parameters set, which are displayed in the respective columns on the **Job Scheduler** page.

Note: Scroll to the right to view the **Actions** column.

5. To enable/disable the tasks, from the **Actions** column, use the **Enable/Disable** toggle.
6. To trigger any of the tasks immediately, from the **Actions** column, click the **TriggerNow**.

Email Settings

SIGN+ includes email setting templates for certification action request. You can customize email IDs for each of the certification action requests. Once you have configured the email settings, emails will be automatically sent to the designated email addresses.

To configure the email settings:

1. Go to **(Menu)** > **SIGN+** > **ADMINISTRATION** > **Email Settings**.
   The **Email Settings** page is displayed.
2. Click on the required certification action request.
3. In the **submission**, **level1ApprovalTo**, and **level2ApprovalTo** fields, enter valid email IDs.
   You can customize the field names by clicking the field name and entering your preferred names. Additionally, you can add more fields by clicking the **Add** button from the top-right corner of the **Email Settings** page. If any of the fields are not required, you can remove them by clicking the delete icon.

   Note: You can enter multiple email addresses, separated by commas.

4. Click **Save Changes**.

Self-Service SIGN+: Download Package & Manage Credentials

This provides a self-service model that enables users to independently download the SIGN+ package and manage their credentials without requiring administrative support.

Objectives

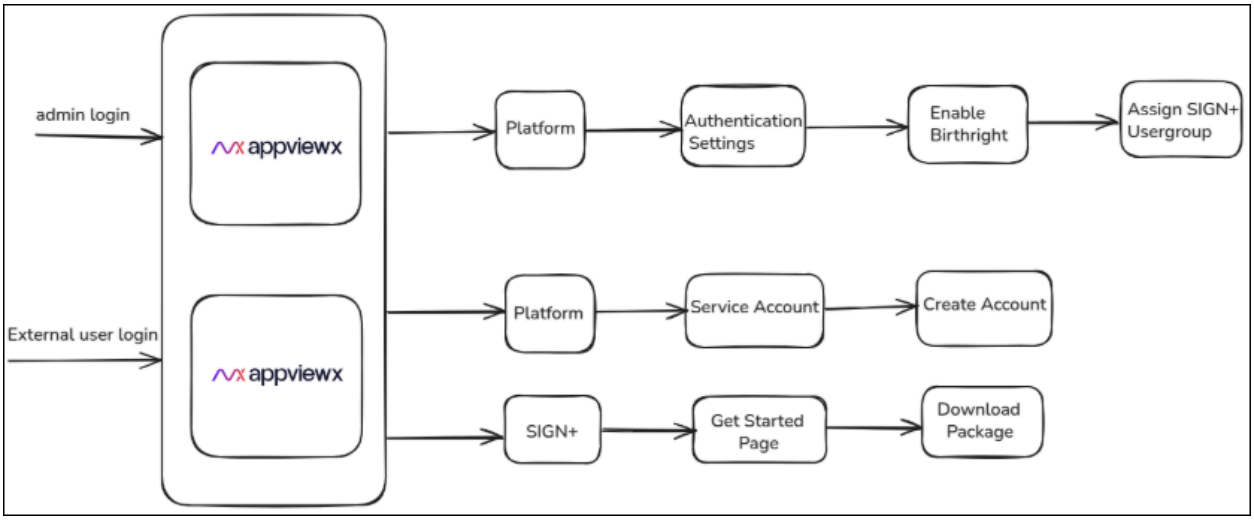
- **Enable user autonomy:** Allow users to access and install the SIGN+ package on their own systems.
- **Credential ownership:** Provide users the ability to generate, view, and manage their own credentials securely.

- **Ensure isolation and security:** Enforce strict separation so that each user can access and manage only their own service accounts and credentials, with no visibility into other users' data.
- **Reduce admin dependency:** Eliminate the need for administrator involvement in distributing the SIGN + package and credential management operations.

Key Requirements

- Secure, authenticated access to download the SIGN+ package.
- Role-based access control **(RBAC)** ensures users can only manage their own accounts and credentials.
- A user-friendly interface to view, create service accounts.
- Enforcement of isolation rules to prevent access to service accounts created by other users.

Self-Service Package Download using SSO with Service Account



Overview

In the proposed model, the administrator creates a dedicated user group within the application for SIGN + users. This group is assigned the required roles and resources, with permissions restricted to SIGN+ functionalities, and is configured in the Authentication settings by enabling birthright access. After setup, external users logging in via SSO are automatically mapped to this group. Once logged in, users can create their own service accounts and associate them with the group. Each service account is securely isolated, visible, and manageable only by its owner. Since SSO credentials cannot be used for API communication with the SIGN+ package, a service account is required for every user.

Steps for the Administrator

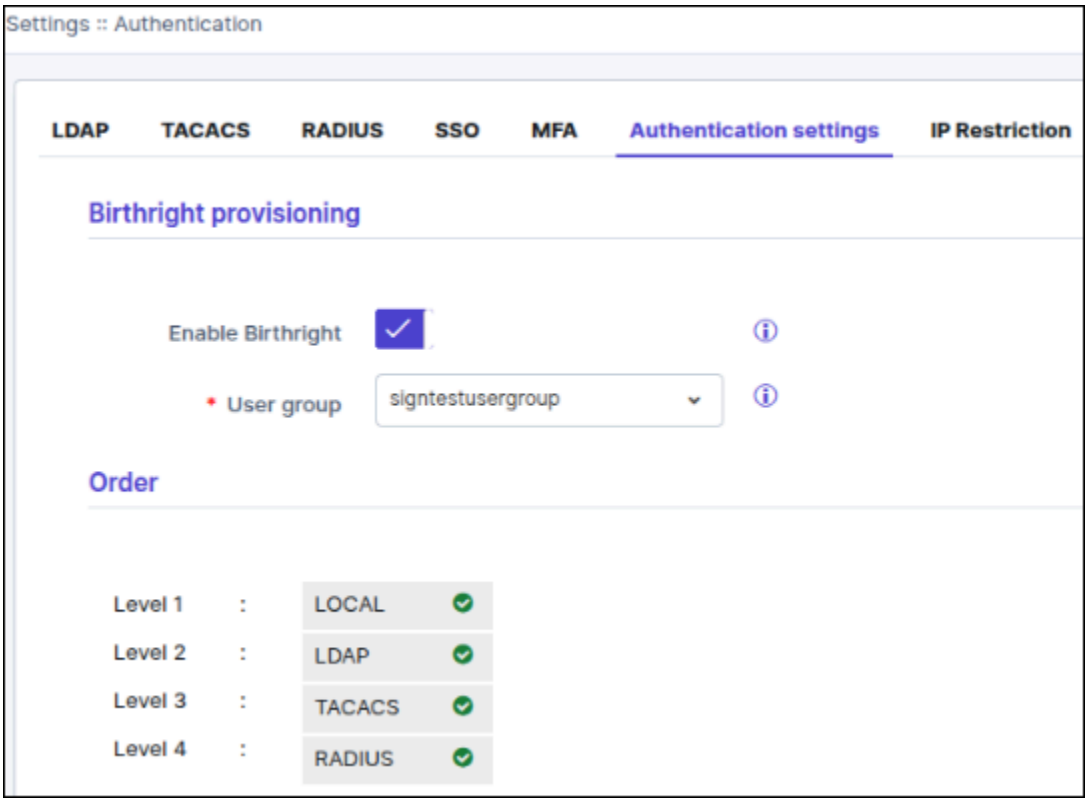
1. The administrator must configure a default user group, which will be associated with a specific Role and Resource containing only the necessary permissions.

   > **Note:** Any user logging in via SSO will automatically be assigned to this user group and will be able to perform only the actions permitted by the assigned role.

2. **Service Account Visibility Control :** The role assigned to a user group will include a crucial setting for managing the visibility of service account details. This setting offers three distinct options, allowing for granular control over who can view information about the service accounts, particularly those created by users within that group.
   - **Self:** When "Self" is selected, users within the assigned group will only be able to view the service account details for accounts they themselves have created. This option ensures a high level of privacy and control, limiting access strictly to the creator of the service account. It's ideal for scenarios where individual accountability and data isolation are paramount.
   - **All:** Choosing "All" will grant users within the assigned group the ability to view the details of all service accounts, regardless of who created them. This option provides complete transparency within the designated user group and is suitable for collaborative environments where multiple team members may need access to various service accounts for operational or troubleshooting purposes.

   The system will dynamically adjust the displayed service accounts based on the permission selected. For example, if "Self" is chosen, the user's interface will only present the service accounts they have personally generated. If "All" is active, a comprehensive list of all service accounts will be visible. This ensures that users consistently have access only to the relevant accounts, maintaining security and preventing unauthorized viewing of sensitive service account information.
3. The administrator needs to configure the SSO authentication server within the platform to enable users to log in via SSO.
   Reference Link: For SSO Integration.
4. The administrator will configure **birthright access** to automatically assign external users (who log in via SSO) to the appropriate user group which was configured in the previous step upon their first login. This setup is managed through the **Platform → Access Management → Authentication Settings** section of the application. By enabling birthright access, the platform ensures that external users are seamlessly mapped to predefined user groups without requiring manual intervention. These user groups are configured with specific roles and permissions relevant to their access needs so that users are granted the correct level of access immediately upon logging into AppViewX via SSO. This

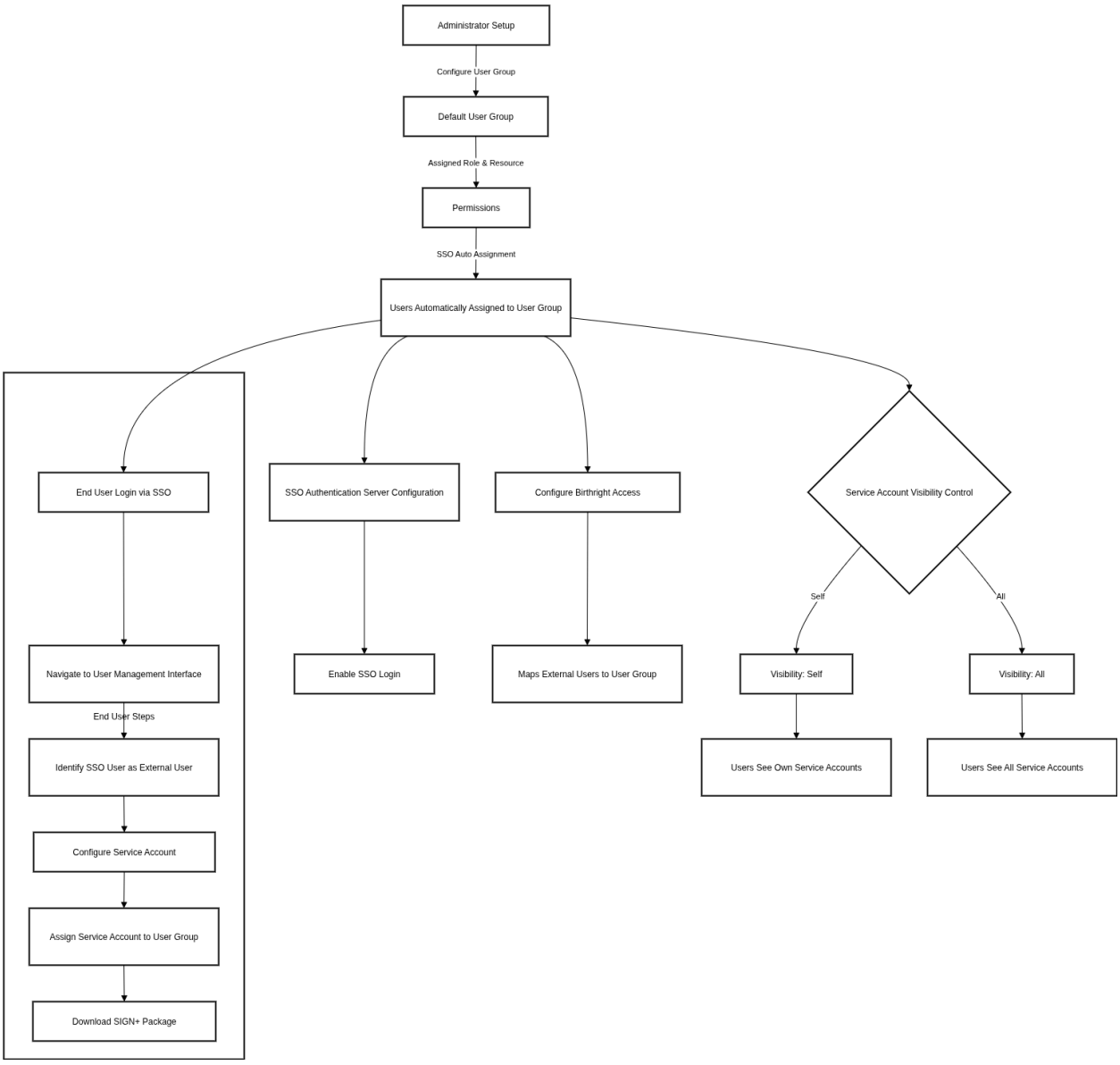
approach streamlines user onboarding while maintaining consistent security and role-based access control.



Steps for the End User

1. To access the application, utilize **Single Sign-On (SSO)** for login. User access within the application's interface is restricted based on the permissions associated with their designated user group.
2. Navigate to **Platform** → **Identity** → **User**. The SSO User will be identified as an external user which would have been assigned to the respective user group which was configured in the **Authentication Settings.**
   Reference Link: Default Roles Creation.
3. Since SSO user credentials do not allow access to **SIGN+** services through **REST** APIs, the user should configure a service account and assign it to the appropriate user group. Navigate to **Platform** → **Identity** → **Service Account** to perform this configuration.
   • Reference Link: Service Account Creation.
   • Reference Link: Service Account Management.
4. The user is now ready to download the **SIGN+** package using the created service account and associated policies to perform signing operations. Navigate to Get started page to download and the SIGN+ Package for the required credential.
   Reference Link: SIGN+ Package Download.

Flowchart on How RBAC Works



Sign Settings

Enabling HSM parallelism previously involved executing database scripts in each environment, often resulting in confusion. To streamline this process, a centralized **Sign Settings** page has been introduced with a global toggle to enable or disable HSM parallelism removing the need for manual script execution. This global configuration of **Number Of Polls** and **Polling Interval**, which are applied as default values to newly created policies. To customize these settings at the policy level, users can enable the new Enable HSM Polling toggle on the **Sign Policy** page, allowing policy-specific values to override the global configuration. This applies only to HSM-based certificates.

Note: This page is accessible only to admin users by default. To grant access to non-admin users, enable the corresponding ACF permission under **Sign+ > Sign Settings**.

To Configure HSM Parallelism in Sign Settings:

1. Go to (**Menu**) > **SIGN+** > **SIGN SETTINGS** > **Sign Settings**.
   The **Signing Settings** page is displayed.
2. If the **Activate HSM Parallel Mode** is disabled, the **Number Of Polls** and **Polling Interval** fields are dispalyed.
3. Enter the ***Number Of Polls**.
   This specifies how many polling attempts should be made for certificates using HSM. The value must be an integer between **1 and 20**.
4. Enter the **Polling Interval**, the time (in milliseconds) between each poll.
   This defines the delay between consecutive polls. The value must be an integer between **1 and 300000** milliseconds.
5. To enable HSM parallelism, select the **Activate HSM Parallel Mode** checkbox.
   When enabled, HSM parallelism handles signing operation statuses internally, improving performance and eliminating the need for manual polling. If HSM devices experience delays, you can disable this mode and configure the global polling settings for more reliable status retrieval.
6. Click **Save**.
   The **Sign Settings** have been saved successfully.

Chapter 3: SIGN+ API Guide

This guide provides information about the AppViewX exposed APIs intended for use in **SIGN+** actions.

API Reference

To try the AppViewX exposed APIs on Swagger, follow these steps:

1. From the top right corner of your screen, click {...} **(API Reference)** icon.

   Note: The **API Reference** icon is displayed for the Admin user and users with ACF permission enabled for API Reference.

   You will be redirected to **AppViewX API Reference** page.

2. Click the required API to **Try it out**.

Enable ACF Permission for API Reference

To enable the ACF permission for non-admin roles to access the API reference, follow these steps:

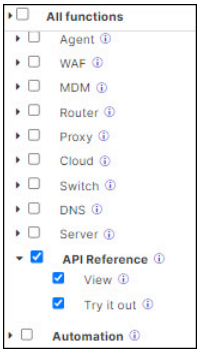
1. Go to **Platform** module **IDENTITY** > **Role**.

   You will be redirected to **Role** page.

2. Click on the role name to enable the ACF permission.

   You will be redirected to the **Modify :: [RoleName]** page, with the **Information** tab open by default.

3. Switch to the **Authorized Functions** tab and expand the **Platform** by clicking **(Expand)** icon.
4. To assign **API Reference**, select the checkbox beside **API Reference**.



5. Click **Save**.

Best Practices for Working with the AppViewX API

- **Use appropriate HTTP methods**

  Ensure that the correct HTTP method is used for each operation (e.g., GET for retrieval, POST for creation).

- **Handle errors gracefully**

  Implement proper error handling in your application to manage API responses.

- **Use secure storage**

  Store access tokens securely and avoid hardcoding them in your application code.

- **Implement pagination**

  For endpoints that return large datasets, implement pagination using limit and offset parameters.

Understanding the AppViewX Sign+ API

The AppViewX SIGN+ API provides a set of RESTful endpoints for managing code signing request across your infrastructure. This section covers how to make requests, handle responses, and understand the structure of the API.

RESTful HTTPS Requests

The SIGN+ API uses RESTful principles, leveraging standard HTTP methods to interact with resources. All requests must be made over HTTPS to ensure security.

Type	Description
GET	GET requests, retrieve resource representation/information only and not to modify it.
POST	POST APIs create new subordinate resources. For example, a file is subordinate to a directory containing it or a row is subordinate to a database table. In terms of REST, POST methods are used to create a new resource into the collection of resources.
PUT	PUT APIs are used to update existing resources (if a resource does not exist then API may decide whether to create a new resource or not).
DELETE	DELETE APIs are used to delete resources (identified by the Request-URI).

Requests

All API endpoints are accessed via the following base URL. The base URL is built in the same way by the following structure:

```
http://<IP/HostName/TenantName>:<GWPORT>/avxapi/<Endpoint>?<gwsource>
```

The explanation has been added to all APIs in the Reference section.

Understanding the sample URL:

- **IP/HostName/TenantName**: Replace with the actual IP address, hostname, or tenant name based on the specific configuration in AppViewX.
  - **IP**: A unique identifier assigned to each device connected to a computer network that uses the Internet Protocol for communication

    The IP address will be included in the endpoint URL for an on-prem deployment.
  - **HostName**: A human-readable label assigned to a device (host) on a network

    The hostname will be included in the endpoint URL for an on-prem deployment.
  - **TenantName**: An identifier label for a tenant given to indicate which tenant's data the API request will access/modify

    The tenant name will be included in the endpoint URL for a SaaS deployment.

- **GWPORT**: AppViewX gateway port

  A gateway port refers to a network port through which data is sent and received to communicate with a gateway in an on-prem deployment.

  Example: **31443**

- **avxapi**: Path parameter value (static) that is part of the endpoint's URL
- **Endpoint**: Endpoint of the API, for example: **execute-hook**
- **gwsource**: Source or origin of a gateway, for example: **external**.

Request Structure

All endpoints accept a request structure that should consist of JSON formatted data. To ensure the request is accepted, set the header **Content-Type: application/json**.

The following example shows a request to add a resource:

```
{
 "payload": {
"name": "resource_1",
"description": "This is a sample resource."
}
}
```

Response Structure

The Content-Type of the response is typically determined by the Content-Type header, and for most endpoints, it will be application/json. All requests that reach the server, regardless of the response code, will retrieve a response body. A successful request will contain a body with the requested information, for example:

```
https://appviewxapi.com/avxapi/resource?gwsource=external
```

Returns the following JSON structure that a resource is added:

```
{
 "response": "Resource added successfully",
 "message": "Resource added successfully",
 "appStatusCode": null,
 "tags": null,
 "headers": null
 }
```

Description of Server Responses

HTTP Code	Response Message
200 OK	The request was successful (some API calls may return 201 or 202 instead).
400 Bad Request	The request is not understood or required parameters are missing.
401 Unauthorized	Authentication failed or the user doesn't have permissions for the requested operation.
409 Forbidden	Access denied.
404 Not Found	Resource not found.
429 Too many requests	The number of requests to the service has crossed the threshold.
503 Service unavailable	The client cannot communicate with the service.
504 Gateway timeout	The given request has exceeded the expected time.

URI Scheme

- **Host :** {url}
- **BasePath :** /avxapi
- **Schemes :** HTTPS
- **URL :** https://{url}/avxapi

Types of Accounts in AppViewX

There are two types of accounts in AppViewX:

- **User Accounts**: These are used by actual users.
- **Service Accounts**: These are used by system services such as web servers, automation tools, and so on.

AppViewX recommends using a Service Account for accessing APIs from automation tools. Service Accounts are supported with oAuth standard for a more secure and standard way of accessing APIs.

Note: AppViewX supports both User Account and Service Account for accessing APIs.

Authentication Using a User Account

For accessing APIs, you can login via two types of accounts:

- User account

  A **User account** represents an individual person interacting with the application or the system. User accounts are used for accessing the system on behalf of a user.

  For accessing APIs with a user account, you need to get the session ID by providing a username and password in the login API. This session ID can then be used for accessing other APIs.

  Note: You can also use the username and password in all API calls instead of the sessionId. However, this is not recommended.

- Retrieve session ID using login API
- Using Session ID for further API calls

Retrieve session ID using login API

This API used to retrieve the session ID using the login API for secure authentication and access to system resources.

Before you begin

- Make sure you have valid login credentials (Username and Password) for accessing the system.
- You cannot use OAuth credentials (Client ID and Client Secret) for login.
- To access the APIs using the service token, use the API with the Service Account.

Request Structure

Endpoint	`/login`
Type	`POST`
Sample URL	`https://<IP/HostName/TenantName>:<GWPORT>/avxapi/login?gwsource=external` To understand the elements of the sample URL, click here.
Content-Type	application/json

Request timeout period	15 minutes

Input Parameters

Name	Description
username *Header*	(Mandatory) Use login name of the user. **Type:** String **Example:** "admin"
password *Header*	(Mandatory) Password for the username. **Type:** String **Example:** "AppViewX@123"
otp *Header*	(Mandatory only if MFA is enabled) If MFA is enabled, enter the OTP received on your registered email ID in the header. Multifactor authentication (MFA) is a security mechanism that requires users to provide two or more verification factors to gain access to a resource If MFA is enabled, and you try to login with only the username and password, you will get the following error upon execution of the API: **MFA is enabled. We have sent an OTP to your email ID: aaa********r@appviewx.com**. In this case, ensure that the OTP is included in the header and try logging in again. **Type:** String **Example:** "OTP : **609700**"
Content-Type *Header*	(Mandatory) The parameter should be set to `application/json` to specify the nature of the data in the payload. **Type:** String **Example:** "application/json"

Input Parameters (continued)

Name	Description
gwsource *Query*	(Mandatory) Source from which the request is triggered. The values can be: • web • external **Type:** String

Response Structure

- **Status Code:** 200 Ok
- **Message:** Login Successful
- **Headers:**
  - **Content-Type:** application/json

Response Parameters

Name	Description
response	The response contains the attributes needed to retrieve the session ID.
message	Success message or failure description in case of error.
appStatusCode	Application specific status code for the response. Will be non-null for failure response.
tags	More info in case of failure response.

Name	Description
status	Indicates the overall status of the response. The values can be: • SUCCESS • FAILURE
appStatusCode	An application-specific status code, if applicable.
statusDescription	Description of the status, if available.
sessionId	Unique identifier for the session.

Name	Description
lockDownPeriod	Number of login attempts remaining.
termsAccepted	
passwordExpiryMsg	
emailId	

Status Codes

HTTP Code	appStatusCode	Response Message
200 OK	NA	Login successful
400 Bad request	ACCT_AUTH_001	Username or password cannot be null or empty.
401 Unauthorized	ACC_AUTH_022	Login failed. Invalid credentials.
401 Unauthorized	ACC_AUTH_006	Login failed. Invalid credentials.

Sample Request/Response

Use Case

Login to the application with a username and password.

Request URL

```
https://<IP/HostName/TenantName>:<GWPORT>/avxapi/login?gwsource=external
```

Request Payload

```
{}
```

Sample Response


```
{
    "response": {
        "status": "SUCCESS",
        "appStatusCode": null,
        "statusDescription": null,
        "sessionId": "avx~~c73a4f56-f4ab-4cdf-aadf-6d90bf406077",
        "authCode": null,
        "lockDownPeriod": 15,
        "emailId": null,
        "termsAccepted": true,
```

```
        "passwordExpiryMsg": ""
    },
    "message": "Login successful.",
    "appStatusCode": null,
    "tags": null,
    "headers": null
}
```


What's Next

- **Using Session ID for further API calls**

Reference

Understanding the sample URL:

- **IP/HostName/TenantName**: Replace with the actual IP address, hostname, or tenant name based on the specific configuration in AppViewX.
  - **IP**: A unique identifier assigned to each device connected to a computer network that uses the Internet Protocol for communication

    The IP address will be included in the endpoint URL for an on-prem deployment.
  - **HostName**: A human-readable label assigned to a device (host) on a network

    The hostname will be included in the endpoint URL for an on-prem deployment.
  - **TenantName**: An identifier label for a tenant given to indicate which tenant's data the API request will access/modify

    The tenant name will be included in the endpoint URL for a SaaS deployment.
- **GWPORT**: AppViewX gateway port

  A gateway port refers to a network port through which data is sent and received to communicate with a gateway in an on-prem deployment.

  Example: **31443**
- **avxapi**: Path parameter value (static) that is part of the endpoint's URL
- **Endpoint**: Endpoint of the API, for example: **execute-hook**
- **gwsource**: Source or origin of a gateway, for example: **external**.

Using Session ID for further API calls

The sessionID retrieved using the login API can be used in the header for making further API calls.

In this section, as an example, we are using the session ID with the API call for adding a role.

Before you begin

- Session ID is obtained from the login API.
- Ensure that the session ID is valid and has not expired.

Request Structure

Endpoint:	`/role`
Type:	`POST`
Sample URL:	`https://<IP/HostName/TenantName>:<GWPORT>/avxapi/role?&gwsource=external` To understand the elements of the sample URL, click here.
Content-Type:	application/json

Input Parameters

Name	Description
sessionId *Header*	(Mandatory) Use session ID retrieved from login API, if username and password are not provided. **Type:** *String* **Example:** "sessionId": "ce7f1a14-2bf9-4e4a-89a8-bc780a255813"
gwsource *Query*	(Mandatory) Source from which the request is triggered. The values can be: • web • external **Type:** *String*
Payload *String*	(Mandatory) Input data for request body in application/json format. For payload details, see **Payload** section.

Payload

Name	Description
name	(Mandatory) Name of the role to be added.
String	**Example:** "role_1"
description	(Optional) Description of the role to be added.
String	**Example:** "Adding a new role"

Response Structure

- **Status Code:** 201 Created
- **Message:** Role added successfully
- **Headers:**
  - **Content-Type:** application/json

Response Parameters

Name	Description
response	Contains the response attributes for role added successfully.
message	Success message or failure description in case of error.
appStatusCode	Application specific status code for the response. Will be non-null for failure response.
tags	More info in case of failure response.

Status Codes

HTTP Code	appStatusCode	Response Message
201 Created	null	Role added successfully.
409 Conflict	ACCT_RO_002	Role name already exists
400 Bad Request	VALIDATION_ERROR_0004	'name' should have at least '2' characters, Mandatory Field 'name' is missing or empty.
400 Bad Request	ACCT_RO_015	Role name invalid.

Sample Request/Response

Use Case

Using the session ID acquired from the login API to execute subsequent API calls, specifically for adding a role API.

Sample Request

```
https://<IP/HostName/TenantName>:<GWPORT>/avxapi/role?gwsource=external
```

Request Payload

```
{
"payload": {
 "name":"role_01",
 "description":"Adding a new role"
}
}
```

Sample Response

```
{
"response": "Role added successfully",
"message": "Role added successfully",
"appStatusCode": null,
"tags": null,
"headers": null
}
```

Reference

Understanding the sample URL:

- **IP/HostName/TenantName**: Replace with the actual IP address, hostname, or tenant name based on the specific configuration in AppViewX.
  - **IP**: A unique identifier assigned to each device connected to a computer network that uses the Internet Protocol for communication

    The IP address will be included in the endpoint URL for an on-prem deployment.
  - **HostName**: A human-readable label assigned to a device (host) on a network

    The hostname will be included in the endpoint URL for an on-prem deployment.

- **TenantName**: An identifier label for a tenant given to indicate which tenant's data the API request will access/modify

  The tenant name will be included in the endpoint URL for a SaaS deployment.

- **GWPORT**: AppViewX gateway port

  A gateway port refers to a network port through which data is sent and received to communicate with a gateway in an on-prem deployment.

  Example: **31443**

- **avxapi**: Path parameter value (static) that is part of the endpoint's URL
- **Endpoint**: Endpoint of the API, for example: **execute-hook**
- **gwsource**: Source or origin of a gateway, for example: **external**.

Authentication Using a Service Account

For accessing APIs, you can login via two types of accounts:

- Service account

  A **Service account** represents a non-human entity such as an application or a service. It is used for automated processes or system-to-system interactions without human intervention.

  For accessing APIs with a service account, you need to get the Access Token by providing Client ID and Client Secret in get-service-token API. This Access Token can then be used for accessing other APIs.

  > **Note:** Access Token Validity is 30 minutes by default and it can be configured in **Settings** > **Authentication** > **oAuth Settings**.

- Retrieve Access Token using get-service-token API
- Using Access Token in the header for further API calls

Retrieve Access Token using get-service-token API

The API provides a streamlined process for retrieving service tokens related to account management tasks.

Before you begin

• Make sure you have valid login credentials for accessing the system.

Request Structure

Endpoint:	`/acctmgmt-get-service-token`
Type:	`POST`
Sample URL:	`https://<IP/HostName/TenantName>:<GWPORT>/avxapi/acctmgmt-get-service-token?gwsource=external` To understand the elements of the sample URL, click here.
Content-Type:	application/json
Authentication:	Yes
Request timeout period	15 minutes

Input Parameters

	Description
Authorization *Header*	(Mandatory) Please form a string in this format <Client ID>:<Client Secret> and do base64 encoding. Then prepend a key 'Basic' before the encoded value. Final value should be "Basic <EncodedValue>". **Type:** *String* **Example:** "admin"
Content-Type *Header*	(Mandatory) The parameter should be set to `application/json` to specify the nature of the data in the payload. **Type:** *String* **Example:** "application/json"
grant_type *Payload*	(Mandatory) Payload Type should be "Form". The value of the param should be "Client_Credentials". **Type:** *Text*

Response Structure

- **Status Code:** 200 Ok
- **Message:** Successful
- **Headers:**
  - **Content-Type:** application/json

Response Parameters

Name	Description
response	The response contains the attributes needed to retrieve the access token.
message	Success message or failure description in case of error.
appStatusCode	Application specific status code for the response. Will be non-null for failure response.
tags	More info in case of failure response.

Status Codes

HTTP Code	appStatusCode	Response Message
200 OK	NA	Successful
400 Bad request	ACCT_SA_003	Service account is invalid/not found::[Service account not found in the database]
400 Bad request	OAUTH_CLNT_015	Client Password is incorrect::[Invalid Client credential]
400 Bad request	ACCT_SA_001	Invalid Request::[Invalid client Id or secret]
500 Internal Server Error	avx-common-011	Error while processing.

Sample Request/Response

Use Case

Retrieve Access Token.

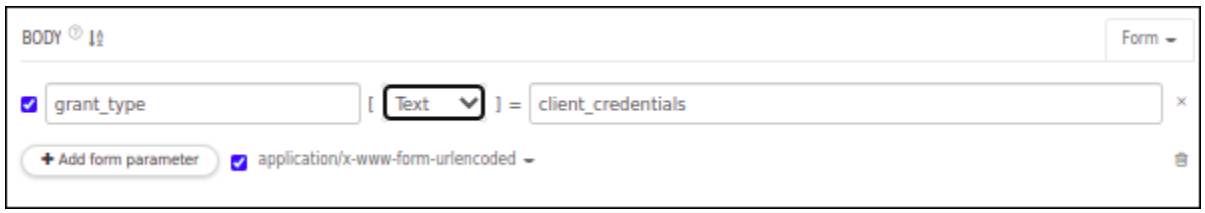
Request URL

```
https://<IP/HostName/TenantName>:<GWPORT>/avxapi/acctmgmt-get-service-token?gwsource=external
```

Content-Type: application/x-www-form-urlencoded

Authorization: Basic
NTIxYzNhZDItZWE0ZS00NDdiLWE1MWItOTYyMWJiN2VhMTI2OjU1QVUjTk84JSpaaGd2TmZhWVVtdHZYMGRrRWhvZ

Request Payload



Sample Response


```
{
"response":
 "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJwbGF0Zm9ybSIsImF1ZCI6ImF2eCIsImNsaWVudElkIjoiOTcwNzRlNDEtOGFmOS00NTZkLTlhNjQtZjBj
NGJiOTA4MDQ4IiwiaXNzIjoiYXZ4IiwiZXhwIjoxNjUwMzY5MzY3LCJncmFudCB0eXBlIjoiY2xpZW50X2NyZWRlbnRpYWxzIn0.HZnkuUEjXIeqJWqpqiNWFHqIDI7
GYf4cWx6VwbjGD_0",
"message": null,
"appStatusCode": null,
"tags":{},
"headers": null
}
```


What's Next

- **Using Access Token in the header for further API calls**

Reference

Understanding the sample URL:

- **IP/HostName/TenantName**: Replace with the actual IP address, hostname, or tenant name based on the specific configuration in AppViewX.
  - **IP**: A unique identifier assigned to each device connected to a computer network that uses the Internet Protocol for communication

    The IP address will be included in the endpoint URL for an on-prem deployment.
  - **HostName**: A human-readable label assigned to a device (host) on a network

The hostname will be included in the endpoint URL for an on-prem deployment.

- **TenantName**: An identifier label for a tenant given to indicate which tenant's data the API request will access/modify

  The tenant name will be included in the endpoint URL for a SaaS deployment.

- **GWPORT**: AppViewX gateway port

  A gateway port refers to a network port through which data is sent and received to communicate with a gateway in an on-prem deployment.

  Example: **31443**

- **avxapi**: Path parameter value (static) that is part of the endpoint's URL
- **Endpoint**: Endpoint of the API, for example: **execute-hook**
- **gwsource**: Source or origin of a gateway, for example: **external**.

Using Access Token in the header for further API calls

The access token retrieved using the get-service-token API can be used in the header for making further API calls.

In this section, as an example, we are using the access token with the API call for adding a resource.

Before you begin

- Access Token is obtained from the get-service-token API.
- Ensure that the Access Token is valid and has not expired.

Request Structure

Endpoint:	`/resource`
Type:	`POST`
Sample URL:	`https://<IP/HostName/TenantName>:<GWPORT>/avxapi/resource?gwsource=external` To understand the elements of the sample URL, click here.
Content-Type:	application/json

Input Parameters

Name	Description
Token *Header*	(Mandatory) Use token retrieved from login API. **Type:** *String* **Example:** eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJwbGF0Zm9ybSIsImF1ZCI6ImF2eCIsImNsaWVudElkIjoiOTcwNzRlNDEtOGFmOS00NTZkLTlhNjQtZjBjNGJiOTA4MDQ4IiwiaXNzIjoiYXZ4IiwiZXhwIjoxNjUwMzY5MzY3LCJncmFudCB0eXBlIjoiY2xpZW50X2NyZWRlbnRpYWxzIn0.HZnkuUEjXIeqJWqpqiNWFHqIDI7GYf4cWx 6VwbjGD_0
gwsource *Query*	(Mandatory) Source from which the request is triggered. The values can be: • web • external **Type:** *String*
Payload *String*	(Mandatory) Input data for request body in application/json format. For payload details, see **Payload** section.

Payload

Name	Description
name *String*	(Mandatory) Name of the resource to create. Name cannot be duplicated. **Example:** "resource_1"
description *String*	(Optional) Description of the resource. **Example:** "This is a sample resource."

Response Structure

- **Status Code:** 201 Created
- **Message:** Resource added successfully
- **Headers:**
  - **Content-Type:** application/json

Response Parameters

Name	Description
response	Contains the response attributes for resource added successfully.
message	Success message or failure description in case of error.
appStatusCode	Application specific status code for the response. Will be non-null for failure response.
tags	More info in case of failure response.

Status Codes

HTTP Code	appStatusCode	Response Message
201 Created	null	Resource added successfully
409 Conflict	RBAC_RE_005	Resource with the given name already exists
400 Bad Request	VALIDATION_ERROR_0004	'name' should have at least '2' characters, Mandatory Field 'name' is missing or empty
400 Bad Request	VALIDATION_ERROR_0004	Invalid "name".
401 Unauthorized	AVX_GW_012	Unauthorized access, reason - Invalid Token
407 Proxy Authentication Required	AVX_GW_011	Session validation failed, reason - Session information is missing.

Sample Request/Response

Use Case

Add a resource using API with Access Token.

Sample Request

```
https://<IP/HostName/TenantName>:<GWPORT>/avxapi/resource?gwsource=external
```

Request Payload

```
{
 "payload": {
```

```
"name": "resource_1",
"description": "This is a sample resource."
}
}
```


Sample Response


```
{
"response": "Resource added successfully",
"message": "Resource added successfully",
"appStatusCode": null,
"tags": null,
"headers": null
}
```


Reference

Understanding the sample URL:

- **IP/HostName/TenantName**: Replace with the actual IP address, hostname, or tenant name based on the specific configuration in AppViewX.
  - **IP**: A unique identifier assigned to each device connected to a computer network that uses the Internet Protocol for communication

    The IP address will be included in the endpoint URL for an on-prem deployment.
  - **HostName**: A human-readable label assigned to a device (host) on a network

    The hostname will be included in the endpoint URL for an on-prem deployment.
  - **TenantName**: An identifier label for a tenant given to indicate which tenant's data the API request will access/modify

    The tenant name will be included in the endpoint URL for a SaaS deployment.
- **GWPORT**: AppViewX gateway port

  A gateway port refers to a network port through which data is sent and received to communicate with a gateway in an on-prem deployment.

  Example: **31443**
- **avxapi**: Path parameter value (static) that is part of the endpoint's URL

- **Endpoint**: Endpoint of the API, for example: **execute-hook**
- **gwsource**: Source or origin of a gateway, for example: **external**.

Code Signing Get Policy

The "code-signing-get-policy" API allows users to retrieve the code signing policy associated with their account. By making a GET request to this endpoint, users can access detailed information about the configured signing policy applied to the code signing process.

Before you begin

- Configure the signing policy with relevant details, ensuring mapping to the enrolled certificate (also identified as the signing key on the signing policy page).

Request Structure

Endpoint:	`/code-signing-get-policy`
Type:	`POST`
Sample URL:	`https://<IP/HostName/TenantName>:<GWPORT>/avxapi/code-signing-get-policy?gwsource=external` To understand the elements of the sample URL, click here.
Content-Type:	application/json

Input Parameter

Name	Description
sessionId *Header*	(Mandatory) After successfully logging in, a unique identifier assigned to a user's session after successful authentication. The session ID remains valid until it expires. The session ID is a string value. **Example:** "ce7f1a14-2bf9-4e4a-89a8-bc780a255813"
username *Header*	(Mandatory) AppViewX login username, represented as a string value. **Example:** "User"
password *Header*	(Mandatory) AppViewX login username, represented as a string value. **Example:** "AppViewX@123"

Input Parameter (continued)

Name	Description
Payload	(Mandatory) Input data for request body in application/json format. For payload details, see **Payload** section.

Payload

Name	Description
skip	(Optional) This field in the payload is used for pagination.
Integer	**Example:** 0
limit	(Optional) This field in the payload is used for pagination.
Integer	**Example:** 25

Response Structure

- **Status Code:** 200 OK
- **Message:** null
- **Headers:**
  - **Content-Type:** application/json

Response Parameters

Name	Description
response	Contains the response attributes for the get policy request.
message	Success message or failure description in case of error.
appStatusCode	Application specific status code for the response. Will be non-null for failure response.
tags	More info in case of failure response.

Status Codes

HTTP Code	appStatusCode	Response Message
200 OK	-	null

HTTP Code	appStatusCode	Response Message
400 Bad Request	avx-common-028	Invalid/ Incorrect payload.
401 Unauthorized	AVX_GW_003	Authentication failed, reason - Invalid Credentials.

Sample Request/Response

Use Case

This API is designed to retrieve comprehensive information about a configured signing policy.

Request URL

```
https://<IP/HostName/TenantName>:<GWPORT>/avxapi/code-signing-get-policy?gwsource=external
```

Request Payload

```
{
  "payload": {
    "skip": 0,
    "limit": 25
  }
}
```

Sample Response

```
{
  "response": {
    "data": [
      {
        "policyName": "FileBasedSigning",
        "fileTypes": [
          "JAR",
          "APK",
          "PS1",
          "EXE",
          "CAB"
        ],
        "restrictionType": "None",
        "ip": null,
        "ipRange": null,
```

```
"signingHashAlgorithm": "SHA-256",
"timeStampingAuthority": "Global Sign",
"timeStampingURL": "",
"status": "Active",
"policyKeyId": "65c49ae81112f940dab1cb31",
"policyMetaInfoId": "65c9ca7ca245650b1ecc75d9",
"permissions": [
  "testsignuser:RW",
  "harshithuser:R",
  "super access:R",
  "super access:RW"
],
"aclIdentifiers": [
  "super access",
  "harshithuser",
  "testsignuser"
],
"signingType": "File Based Signing",
"createdDate": 1711958003401,
"keywords": [
  "FileBasedSigning",
  "Global Sign",
  "Active"
],
"testPolicy": false,
"emailNotification": true,
"subject": "Test Email",
"toEmailList": [
  "jayaharshith.ambati@appviewx.com"
],
"event": "Both",
"requiredFields": [
  {
    "label": "Policy Name",
    "value": "policyName"
  },
  {
```

```
      "label": "Key Name",
      "value": "keyName"
    },
    {
      "label": "IP Address",
      "value": "ipAddress"
    },
    {
      "label": "Signing Time",
      "value": "signingTime"
    },
    {
      "label": "Username",
      "value": "username"
    },
    {
      "label": "Signing Type",
      "value": "signingType"
    }
  ],
  "noOfPolls": null,
  "pollingInterval": null,
  "_id": "65c49aee1112f940dab1cb32"
},
{
  "policyName": "HashPolicy_Test",
  "fileTypes": null,
  "restrictionType": "None",
  "ip": null,
  "ipRange": null,
  "signingHashAlgorithm": "SHA-256",
  "timeStampingAuthority": "Entrust",
  "timeStampingURL": "",
  "status": "Active",
  "policyKeyId": "65d45b1ca245650b1ecc7632",
  "policyMetaInfoId": "66015dbdb498e701acbaf0c3",
  "permissions": [
```

```
      "harshithuser:R",
      "super access:RW"
    ],
    "aclIdentifiers": [
      "super access",
      "harshithuser"
    ],
    "signingType": "Hash Based Signing",
    "createdDate": 1712059613972,
    "keywords": [
      "HashPolicy_Test",
      "Entrust",
      "Active"
    ],
    "testPolicy": false,
    "emailNotification": false,
    "subject": null,
    "toEmailList": null,
    "event": null,
    "requiredFields": null,
    "noOfPolls": 5,
    "pollingInterval": 10,
    "_id": "65d45b2ca245650b1ecc7633"
  },
  {
    "policyName": "TestPolicy1",
    "fileTypes": null,
    "restrictionType": "None",
    "ip": null,
    "ipRange": null,
    "signingHashAlgorithm": "SHA-256",
    "timeStampingAuthority": "Symantec",
    "timeStampingURL": "",
    "status": "Active",
    "policyKeyId": "65eeb58929b67031c341084c",
    "policyMetaInfoId": "",
    "permissions": [
```

```
        "harshithuser:R",
        "super access:RW"
      ],
      "aclIdentifiers": [
        "super access",
        "harshithuser"
      ],
      "signingType": "Hash Based Signing",
      "createdDate": 1710143404600,
      "keywords": [
        "TestPolicy1",
        "Symantec",
        "Active"
      ],
      "testPolicy": false,
      "emailNotification": false,
      "subject": null,
      "toEmailList": null,
      "event": null,
      "requiredFields": null,
      "noOfPolls": 5,
      "pollingInterval": 10,
      "_id": "65eeb7ac29b67031c341084d"
    }
  ],
  "iTotalDisplayRecords": 3,
  "totalCount": 0
},
"message": null,
"appStatusCode": null,
"tags": null,
"headers": null
}
```


What's Next

- **Code Signing with Upload & Sign**
- **Generate Hash for Code Signing**.

Reference

Understanding the sample URL:

- **IP/HostName/TenantName**: Replace with the actual IP address, hostname, or tenant name based on the specific configuration in AppViewX.
  - **IP**: A unique identifier assigned to each device connected to a computer network that uses the Internet Protocol for communication

    The IP address will be included in the endpoint URL for an on-prem deployment.
  - **HostName**: A human-readable label assigned to a device (host) on a network

    The hostname will be included in the endpoint URL for an on-prem deployment.
  - **TenantName**: An identifier label for a tenant given to indicate which tenant's data the API request will access/modify

    The tenant name will be included in the endpoint URL for a SaaS deployment.
- **GWPORT**: AppViewX gateway port

  A gateway port refers to a network port through which data is sent and received to communicate with a gateway in an on-prem deployment.

  Example: **31443**
- **avxapi**: Path parameter value (static) that is part of the endpoint's URL
- **Endpoint**: Endpoint of the API, for example: **execute-hook**
- **gwsource**: Source or origin of a gateway, for example: **external**.

Code Signing with Upload & Sign

This API is used for code signing with upload and sign. It establishes the policies and permissions that oversee the process of uploading and signing code files. Its primary purpose is to ensure a secure and authorized code signing process, playing a crucial role in preserving control and compliance throughout code deployment and execution.

Before you begin

- Configure the signing policy with relevant details, ensuring mapping to the enrolled certificate (also identified as the signing key on the signing policy page).
- The file types selected during policy creation are the only ones permitted for upload. Supported file types include: PS1, EXE, CAT, MSI, JS, JAR, APK, VBS, CAB, WSF, DLL, PSM1, PSD1, PS1XML, JSE, and VBE.

Request Structure

Endpoint:	`/code-signing-upload-sign-file-policy`
Type:	`POST`
Sample URL:	`https://<IP/HostName/TenantName>:<GWPORT>/avxapi/code-signing-upload-sign-file-policy?gwsource=external` To understand the elements of the sample URL, click here.
Content-Type:	application/json

Input Parameter

Name	Description
sessionId *Header*	(Mandatory) After successfully logging in, a unique identifier assigned to a user's session after successful authentication. The session ID remains valid until it expires. The session ID is a string value. **Example:** "ce7f1a14-2bf9-4e4a-89a8-bc780a255813"
username *Header*	(Mandatory) AppViewX login username, represented as a string value. **Example:** "User"
password *Header*	(Mandatory) AppViewX login username, represented as a string value. **Example:** "AppViewX@123"
Payload	(Mandatory) Input data for request body in application/json format. For payload details, see **Payload** section.

Payload

Name	Description
file *binary*	(Mandatory) Upload the file for code signing. **Example:** "binary"
fileName *String*	(Mandatory) Name of the file which is a string value. **Example:** "AppViewX.jar"
fileType *String*	(Mandatory) Specific format of a file providing essential metadata for proper handling and processing which is a string value. **Example:** "JAR"
signingPolicy *String*	(Mandatory) Enter the signing policy for code signing which is a string value. **Example:** "testPolicyByAppViewX"
signingKey *String*	(Mandatory) Enter the signing key for code signing which is a string value. **Example:** "GCA_CSP_Cert=E8:F1:1A:04:29:BF:72:44:85:2A:18:12:70:5F:74:F6:42:79:CA"
signedType *String*	(Mandatory) Select the code signed type, a string that specifies File Based sign. **Example:** "File Based Signing"
signatureType *String*	(Optional) This ensures compliance with a designated signature format while also allowing for potential support of additional signing types in the future. **Example:** "RAW"
addOnFields *List<Map<String, String>>*	(Optional) Specify additional fields needed for code signing. **Example:** "addOnFields": [{"Version":"V1"},{"Build":"1"}]

Response Structure

- **Status Code:** 200 OK
- **Message:** Successful

- **Headers:**
  - **Content-Type:** application/json

Response Parameters

Name	Description
response	Contains the response attributes for the upload and sign request.
message	Success message or failure description in case of error.
appStatusCode	Application specific status code for the response. Will be non-null for failure response.
tags	More info in case of failure response.

Status Codes

HTTP Code	appStatusCode	Response Message
200 OK	null	Successful
400 Bad Request	CODE_SIGNING_0080	Wrong Input Payload for the text fields in the text block
400 Bad Request	CODE_SIGNING_0081	Invalid Number added in the Add-ons section
400 Bad Request	CODE_SIGNING_0082	Mandatory fields are missing in the Add-ons Section
500 Internal Server Error	CODE_SIGNING_0063	Your chosen signing type is not supported by the selected policy
403 Forbidden	CODE_SIGNING_0058	Unsupported file type is uploaded. The policy selected doesn't support uploaded file type
500 Internal Server Error	CODE_SIGNING_0062	Ip provided is invalid
403 Forbidden	CODE_SIGNING_0031	Permissions are not there to upload file for signing
500 Internal Server Error	CODE_SIGNING_0070	Signing Key is not mapped to the given policy.
500 Internal Server Error	CODE_SIGNING_0073	Certificate is not present in the cert inventory

HTTP Code	appStatusCode	Response Message
500 Internal Server Error	CODE_SIGNING_0087	Signing Key is Revoked/Expired
500 Internal Server Error	CODE_SIGNING_0020	Error in generating the signed file
500 Internal Server Error	CODE_SIGNING_0023	I/O Exception occurred
500 Internal Server Error	CODE_SIGNING_0022	Error in generating the signature file
500 Internal Server Error	CODE_SIGNING_0021	Error in updating the signed data
500 Internal Server Error	CODE_SIGNING_00220	Your chosen signature type is currently not supported.

Sample Request/Response

Use Case

To sign a file using **code-signing-upload-sign-file-policy** API.

Request URL

```
https://<IP/HostName/TenantName>:<GWPORT>/avxapi/code-signing-upload-sign-file-policy?gwsource=external
```

Request Payload

```
{
"payload" : {
file: (binary)
fileName: AppViewX.jar
fileType: JAR
signingPolicy: testPolicyByAppViewX
signingKey: GCA_CSP_Cert=E8:F1:1A:04:29:BF:72:44:85:2A:18:12:70:5F:74:F6:42:79:CA
signedType: File Based Signing
signatureType: RAW
addOnFields: [{"Version":"V1"},{"Build":"1"}]
   }
}
```

Sample Response

```
{
   "response": "65252c675e3734782705b4cd",
   "message": null,
   "appStatusCode": null,
   "tags": null,
   "headers": null
}
```


What's Next

- **Fetching the status of the signing request**
- **Download Code Signed File**.

Reference

Understanding the sample URL:

- **IP/HostName/TenantName**: Replace with the actual IP address, hostname, or tenant name based on the specific configuration in AppViewX.
  - **IP**: A unique identifier assigned to each device connected to a computer network that uses the Internet Protocol for communication

    The IP address will be included in the endpoint URL for an on-prem deployment.
  - **HostName**: A human-readable label assigned to a device (host) on a network

    The hostname will be included in the endpoint URL for an on-prem deployment.
  - **TenantName**: An identifier label for a tenant given to indicate which tenant's data the API request will access/modify

    The tenant name will be included in the endpoint URL for a SaaS deployment.
- **GWPORT**: AppViewX gateway port

  A gateway port refers to a network port through which data is sent and received to communicate with a gateway in an on-prem deployment.

  Example: **31443**
- **avxapi**: Path parameter value (static) that is part of the endpoint's URL
- **Endpoint**: Endpoint of the API, for example: **execute-hook**
- **gwsource**: Source or origin of a gateway, for example: **external**.

Fetching the status of the signing request

The API allows users to retrieve the number of inprogress requests, the current status of their signing requests, and also to let users know if the request has failed due to a timeout error.

Before you begin

- Configure the signing policy with relevant details, ensuring mapping to the enrolled certificate (also identified as the signing key on the signing policy page).
- Make sure you have the Sign ID of the signing request for which you intend to check the status.

Request Structure

Endpoint:	`/code-signing-fetch-status-sync-requests`
Type:	`GET`
Sample URL:	`https://<IP/HostName/TenantName>:<GWPORT>/avxapi/code-signing-fetch-status-sync-requests?gwsource=api&signId=<signId>` To understand the elements of the sample URL, click here.
Content-Type:	application/json

Input Parameters

Name	Description
sessionId *Header*	(Mandatory) After successfully logging in, a unique identifier assigned to a user's session after successful authentication. The session ID remains valid until it expires. The session ID is a string value. **Example:** "a1b2c3d4e5f6"
username *Header*	(Mandatory) AppViewX login username, represented as a string value. **Example:** "User"
password *Header*	(Mandatory) AppViewX login username, represented as a string value. **Example:** "AppViewX@123"
signId	(Mandatory) Enter the Sign ID received after signing the code.

Query Params

Input Parameters (continued)

Name	Description
String	**Example:** "65c47fa41112f940dab1cb12"

Response Structure

- **Status Code:** 200 OK
- **Message:** Successful
- **Headers:**
  - **Content-Type:** application/json

Response Parameters

Name	Description
response	Contains the response attributes for the fetch status sync requests.
message	Success message or failure description in case of error.
appStatusCode	Application specific status code for the response. Will be non-null for failure response.
tags	More info in case of failure response.

Status Codes

HTTP Code	appStatusCode	Response Message
200 OK	null	Successful
400 Bad Request	CODE_SIGNING_0105	SIGN_ID Not present.
400 Bad Request	CODE_SIGNING_0102	Data not present for given signId.
403 Forbidden	CODE_SIGNING_00109	Permissions are not there to fetch status for given signId.
500 Internal Server Error	CODE_SIGNING_0101	Error in Fetching the status for the requested SignId.
500 Internal Server Error	CODE_SIGNING_0103	Fetching Encoded Sign Data Failed.

Sample Request/Response

Use Case

To fetch the status of the signing request.

Request URL

```
https://<IP/HostName/TenantName>:<GWPORT>/avxapi/code-signing-fetch-status-sync-requests?gwsource=api&signId=<signId>
```

Request Payload

```
NA
```

Sample Response 1


```
{
    "response": {
        "status": "Inprogress",
        "noOfInProgressRequests": 1,
        "failedDueToTimeoutError": false
    },
    "message": null,
    "appStatusCode": null,
    "tags": null,
    "headers": null
}
```


Sample Response 2


```
{
"response": {
"encodedHashData": "Get-PSDrive\r\n# SIG # Begin signature block\r\n#
 MIIWUAYJKoZIhvcNAQcCoIIWQTCCFj0CAQExDzANBglghkgBZQMEAgEFADB5Bgor\r\n#
 YXQNcC2NQme7lajGfHWbGBOT9EyB/78Wv2/i/GgcblLUPrd/7I7yOi4sITChar8J\r\n#
 iJO1GbYJjzUMAhGb64sD4jlQnRj69hWKvG5uy/yu5OD39F1WvVCvTOT7FaR/HQIN\r\n# V9orkA==\r\n# SIG # End signature block\r\n",
"status": "Signed"
},
"message": null,
"appStatusCode": null,
"tags": null,
"headers": null
}
```

What's Next

- **Download Code Signed File**.

Reference

Understanding the sample URL:

- **IP/HostName/TenantName**: Replace with the actual IP address, hostname, or tenant name based on the specific configuration in AppViewX.
  - **IP**: A unique identifier assigned to each device connected to a computer network that uses the Internet Protocol for communication

    The IP address will be included in the endpoint URL for an on-prem deployment.
  - **HostName**: A human-readable label assigned to a device (host) on a network

    The hostname will be included in the endpoint URL for an on-prem deployment.
  - **TenantName**: An identifier label for a tenant given to indicate which tenant's data the API request will access/modify

    The tenant name will be included in the endpoint URL for a SaaS deployment.
- **GWPORT**: AppViewX gateway port

  A gateway port refers to a network port through which data is sent and received to communicate with a gateway in an on-prem deployment.

  Example: **31443**
- **avxapi**: Path parameter value (static) that is part of the endpoint's URL
- **Endpoint**: Endpoint of the API, for example: **execute-hook**
- **gwsource**: Source or origin of a gateway, for example: **external**.

Download Code Signed Files

This API allows users to download code-signed files that have been digitally signed and verified, ensuring the integrity and authenticity of the downloaded content.

Before you begin

- Ensure successful signing of the file from the Signing Inventory
- Ensure you have the Sign ID of the code-signed file that you intend to download.

Request Structure

Endpoint:	`/code-signing-download-signed-file`
Type:	`POST`
Sample URL:	`https://<IP/HostName/TenantName>:<GWPORT>/avxapi/code-signing-download-signed-file?gwsource=external` To understand the elements of the sample URL, click here.
Content-Type:	application/json

Input Parameters

Name	Description
sessionId *Header*	(Mandatory) After successfully logging in, a unique identifier assigned to a user's session after successful authentication. The session ID remains valid until it expires. The session ID is a string value. **Example:** "a1b2c3d4e5f6"
username *Header*	(Mandatory) AppViewX login username, represented as a string value. **Example:** "User"
password *Header*	(Mandatory) AppViewX login username, represented as a string value. **Example:** "AppViewX@123"
Payload	(Mandatory) Input data for request body in application/json format. For payload details, see **Payload** section.

Payload

Name	Description
signId *String* **Payload**	(Mandatory) Enter the Sign ID received after signing the code, which is a string value. **Example:** "651baff382ca812a7cbf4baa"

Response Structure

- **Status Code:** 200 OK
- **Message:** Successful
- **Headers:**
  - **Content-Type:** application/json
- **Response:** Signed File is downloaded.

Status Codes

HTTP Code	appStatusCode	Response Message
200 Ok	null	Successful
400 Payload entered is Invalid.	VALIDATION_ERROR_0004	Input fields do not comply with the validation criteria. Please recheck the input payload::[Id is mandatory]
400 SignId entered is invalid.	VALIDATION_ERROR_0004	Invalid 'signId
403 Invalid permissions.	CODE_SIGNING_0076	Permissions are not there to download the signed file for the given input
500 SignId does not exist.	CODE_SIGNING_0068	Sign Id Does not Exist.
500 Invalid input.	CODE_SIGNING_0069	Download action can not be performed on Hash Based Signing/Failed status entry.
500 Unavailability of resources.	CODE_SIGNING_0014	Download operation failed for the given sign id

Sample Request/Response

Use Case

To download a code signed file using Sign Id.

Request URL

```
https://<IP/HostName/TenantName>:<GWPORT>/avxapi/code-signing-download-signed-file?gwsource=external
```

Request Payload

```
{
Payload : {
signId: "651baff382ca812a7cbf4baa"
}
}
```

Sample Response

```
Signed File
```

Reference

Understanding the sample URL:

- **IP/HostName/TenantName**: Replace with the actual IP address, hostname, or tenant name based on the specific configuration in AppViewX.
  - **IP**: A unique identifier assigned to each device connected to a computer network that uses the Internet Protocol for communication

    The IP address will be included in the endpoint URL for an on-prem deployment.
  - **HostName**: A human-readable label assigned to a device (host) on a network

    The hostname will be included in the endpoint URL for an on-prem deployment.
  - **TenantName**: An identifier label for a tenant given to indicate which tenant's data the API request will access/modify

    The tenant name will be included in the endpoint URL for a SaaS deployment.
- **GWPORT**: AppViewX gateway port

  A gateway port refers to a network port through which data is sent and received to communicate with a gateway in an on-prem deployment.

  Example: **31443**
- **avxapi**: Path parameter value (static) that is part of the endpoint's URL

- **Endpoint**: Endpoint of the API, for example: **execute-hook**
- **gwsource**: Source or origin of a gateway, for example: **external**.

Generate Digital Signature for Hash

This API allows users to compute the digital signature for a given hash using a specified signing policy and key details.

Before you begin

- Configure the signing policy with relevant details, ensuring mapping to the enrolled certificate (also identified as the signing key on the signing policy page).

Request Structure

Endpoint:	`/code-signing-generate-hash`
Type:	`POST`
Sample URL:	`https://<IP/HostName/TenantName>:<GWPORT>/avxapi/code-signing-generate-hash?gwsource=external` To understand the elements of the sample URL, click here.
Content-Type:	application/json

Input Parameter

Name	Description
sessionId *Header*	(Mandatory) After successfully logging in, a unique identifier assigned to a user's session after successful authentication. The session ID remains valid until it expires. The session ID is a string value. **Example:** "ce7f1a14-2bf9-4e4a-89a8-bc780a255813"
username *Header*	(Mandatory) AppViewX login username, represented as a string value. **Example:** "User"
password *Header*	(Mandatory) AppViewX login username, represented as a string value. **Example:** "AppViewX@123"

Input Parameter (continued)

Name	Description
Payload	(Mandatory) Input data for request body in application/json format. For payload details, see **Payload** section.

Payload

Name	Description
signingPolicy *String*	(Mandatory) Enter the signing policy for code signing which is a string value. **Example:** "Test_Policy_01"
signingKey *String*	(Mandatory) Enter the signing key for code signing which is a string value. **Example:** "Google CA Code Signing Certificate_Demo=A5:09:C1:6C:3F:72:81:61:59:3A:58:EA:ED:33:11:ED:64:91:DC"
versionNumber *String*	(Mandatory) Enter the version number for code signing, which should be a string value. **Example:** "v1"
description *String*	(Mandatory) Description of the hash generation, provided as a string value. **Example:** "Hash Signing"
signedType *String*	(Mandatory) Select the code signed type, a string that specifies Hash Based sign. **Example:** "Hash Based Signing"
fileHashContent *String*	(Mandatory) Enter the hash file content as a string value. **Example:** "MDEwDQYJYIZIAWUDBAIBBQAEIPw9hz6RJNKrng4tnsFCUGKXA6qAyxRe2kFVOjdpfTMw"
signatureType *String*	(Optional) This ensures compliance with a designated signature format while also allowing for potential support of additional signing types in the future. **Example:** "RAW"

Name	Description
paddingType *String*	(Optional) Select the padding type based on your file hash content: Use `NonePaddingTypeWithHex` for Hex content, or `NonePaddingTypeWithBase64Encoded` for Base64 encoded content. • `NonePaddingTypeWithHex` **Example:** 6f2a0e801491873cc411c6d35be91127d80bd6c25946d974ce05e2b9f58bd0c4 • `NonePaddingTypeWithBase64Encoded` **Example:** byoOgBSRhzzEEcbTW+kRJ9gL1sJZRtl0zgXiufWL0MQ=
addOnFields *List<Map<String, String>>*	(Optional) Specify additional fields needed for code signing. **Example:** "addOnFields": [{"Version":"V1"},{"Build":"1"}]

Response Structure

- **Status Code:** 200 OK
- **Message:** Successful
- **Headers:**
  - **Content-Type:** application/json

Response Parameters

Name	Description
response	Contains the response attributes for generating the signature for the code signing request.
message	Success message or failure description in case of error.
appStatusCode	Application specific status code for the response. Will be non-null for failure response.
tags	More info in case of failure response.

Status Codes

HTTP Code	appStatusCode	Response Message
200 OK	null	Successful
403 Forbidden	CODE_SIGNING_0032	Permissions are not there to sign the hash of a file
500 Internal Server Error	CODE_SIGNING_0062	IP provided is invalid
500 Internal Server Error	CODE_SIGNING_0083	The retrieved IP address is not valid. The selected policy does not support the obtained IP address.
500 Internal Server Error	CODE_SIGNING_0063	Your chosen signing type is not supported by the selected policy
500 Internal Server Error	CODE_SIGNING_0056	Signing Policy Info is not present in the Database for the given input
500 Internal Server Error	CODE_SIGNING_0070	Signing Key is not mapped to the given policy.
500 Internal Server Error	CODE_SIGNING_0073	Certificate is not present in the cert inventory
500 Internal Server Error	CODE_SIGNING_0021	Error in updating the signed data
500 Internal Server Error	CODE_SIGNING_0066	Failed to convert to json string
500 Internal Server Error	CODE_SIGNING_0067	Failed to encrypt
500 Internal Server Error	CODE_SIGNING_0020	Error in generating the signed file
400 Bad Request	CODE_SIGNING_00222	Add-on fields are not configured for the given policy.
400 Bad Request	CODE_SIGNING_0082	Mandatory fields are missing in the Add-ons Section.
400 Bad Request	CODE_SIGNING_00225	Multiple Add-on fields within a single key-value pair is not allowed.
400 Bad Request	CODE_SIGNING_00223	Provided Add-on fields are not configured for the given policy.

HTTP Code	appStatusCode	Response Message
400 Bad Request	CODE_SIGNING_0080	Wrong Input Payload for the text fields in the text block.
400 Bad Request	CODE_SIGNING_0081	Invalid Number added in the Add-ons section.
500 Internal Server Error	CODE_SIGNING_00220	Your chosen signature type is currently not supported.

Sample Request/Response

Use Case

To generate a hash for code signing using **code-signing-generate-hash** API.

Request URL

```
https://<IP/HostName/TenantName>:<GWPORT>/avxapi/code-signing-generate-hash?gwsource=external
```

Request Payload - 1

```
{
  "payload": {
    "signingPolicy": "Hash_Policy",
    "signingKey": "AppViewX Private Ltd=56:37:33:0E:B1:7D:E4:69:E7:8E:CF:83:56:59:43:93:DD:18:B4",
    "description": "Hash Signing",
    "signedType": "Hash Based Signing",
    "fileHashContent": "MDEwDQYJYIZIAWUDBAIBBQAEIPw9hz6RJNKrng4tnsFCUGKXA6qAyxRe2kFVOjdpfTMw",
    "signatureType": "RAW",
    "addOnFields": [
      {
        "Version": "V1"
      },
      {
        "Build_No": "1"
      }
    ]
  }
}
```

Request Payload - 2

```
{
  "payload": {
```

```
  "signingPolicy": "Hash_Policy",
  "signingKey": "AppViewX Private Ltd=56:37:33:0E:B1:7D:E4:69:E7:8E:CF:83:56:59:43:93:DD:18:B4",
  "description": "Hash Signing",
  "signedType": "Hash Based Signing",
  "paddingType" : "NonePaddingTypeWithHex",
  "fileHashContent": "6f2a0e801491873cc411c6d35be91127d80bd6c25946d974ce05e2b9f58bd0c4",
  "signatureType": "RAW",
  "addOnFields": [
   {
    "Version": "V1"
   },
   {
    "Build_No": "1"
   }
  ]
 }
}
```


Request Payload - 3


```
{
 "payload": {
  "signingPolicy": "Hash_Policy",
  "signingKey": "AppViewX Private Ltd=56:37:33:0E:B1:7D:E4:69:E7:8E:CF:83:56:59:43:93:DD:18:B4",
  "description": "Hash Signing",
  "signedType": "Hash Based Signing",
  "paddingType" : "NonePaddingTypeWithBase64Encoded",
  "fileHashContent": "byoOgBSRhzzEEcbTW+kRJ9gL1sJZRtl0zgXiufWL0MQ=",
  "signatureType": "RAW",
  "addOnFields": [
   {
    "Version": "V1"
   },
   {
    "Build_No": "1"
   }
  ]
 }
}
```

Sample Response


```
{
   "response":
 "gutIcFnlzbTT7slB1wrOAbMPzhgFszs8nA1DpMLE/7BcAP39vbgIOCIj1rlmM6bSnBl1bJ3U3CMSWqphEu8KzN9gcCknGTyAOJxEiIXOmi0P9ernL4knxoGnDe//
89/rC3drt4XqLahHF7mMKrXLCLGqg0UTpOzUM0ZxQTucz4Z2iWipH3R3wNq4gYB4EijPXkp+7D0Q2PGaIiy9/1LhGzwvappbqU9QBFu3Nkr40jepEs7dGcEFYlw4
E1spH+gcJsFEAN1H3UToP6zDiBSEq0ZiwXj0mU+pJGxIG49x7jOaDJgAS+p6/ll9eulwRk7Ft4NXoXwWkvYZTx2HAMz0mg==",
   "message": null,
   "appStatusCode": null,
   "tags": null,
   "headers": null
}
```


Reference

Understanding the sample URL:

- **IP/HostName/TenantName**: Replace with the actual IP address, hostname, or tenant name based on the specific configuration in AppViewX.
  - **IP**: A unique identifier assigned to each device connected to a computer network that uses the Internet Protocol for communication

    The IP address will be included in the endpoint URL for an on-prem deployment.
  - **HostName**: A human-readable label assigned to a device (host) on a network

    The hostname will be included in the endpoint URL for an on-prem deployment.
  - **TenantName**: An identifier label for a tenant given to indicate which tenant's data the API request will access/modify

    The tenant name will be included in the endpoint URL for a SaaS deployment.
- **GWPORT**: AppViewX gateway port

  A gateway port refers to a network port through which data is sent and received to communicate with a gateway in an on-prem deployment.

  Example: **31443**
- **avxapi**: Path parameter value (static) that is part of the endpoint's URL
- **Endpoint**: Endpoint of the API, for example: **execute-hook**
- **gwsource**: Source or origin of a gateway, for example: **external**.

Code Signing Download Certificate

The "Code Signing Download Certificate" API facilitates the retrieval of code signing certificates securely. It enables users to download their code signing certificates.

Before you begin

- Configure the signing policy with relevant details, ensuring mapping to the enrolled certificate (also identified as the signing key on the signing policy page).
- Ensure that you have the necessary payload details of the code signing certificate you intend to download.

Request Structure

Endpoint:	`/code-signing-download-certificate`
Type:	`POST`
Sample URL:	`https://<IP/HostName/TenantName>:<GWPORT>/avxapi/code-signing-download-certificate?gwsource=external` To understand the elements of the sample URL, click here.
Content-Type:	application/json

Input Parameter

Name	Description
Token *String* **(header)**	(Mandatory) Use token retrieved from login API. **Example:** eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJwbGF0Zm9ybSIsImF1ZCI6ImF2eCIsImNsaWVudElkIjoiOTcwNzRlNDEtOGFmOS00NTZkLTlhNjQtZjBjNGJiOTA4MDQ4IiwiaXNzIjoiYXZ4IiwiZXhwIjoxNjUwMzY5MzY3LCJncmFudCB0eXBlIjoiY2xpZW50X2NyZWRlbnRpYWxzIn0.HZnkuUEjXIeqJWqpqiNWFHqIDI7GYf4cWx 6VwbjGD_0
sessionId *String* **(header)**	(Mandatory) After successfully logging in, a unique identifier assigned to a user's session after successful authentication. The session ID remains valid until it expires. The session ID is a string value. **Example:** "ce7f1a14-2bf9-4e4a-89a8-bc780a255813"

Input Parameter (continued)

Name	Description
username *String* **(header)**	(Mandatory) AppViewX login username, represented as a string value. **Example:** "User"
password *String* **(header)**	(Mandatory) AppViewX login username, represented as a string value. **Example:** "AppViewX@123"
gwsource *String* **(query)**	(Mandatory) Source from which the request is triggered. The values can be: • web • external **Type:** *String*
Payload *String*	(Mandatory) Input data for request body in application/json format. For payload details, see **Payload** section.

Payload

Name	Description
commonName *String*	(Mandatory) Enter the common name of the requested certificate. **Example:** "EJBCACertHSM"
serialNumber *String*	(Mandatory) Enter the serial number of the requested certificate. **Example:** "18:C1:CD:90:72:FA:84:5A:87:30:7B:F7:11:47:69:B5:B0:BB:D5:57"
policyName *String*	(Mandatory) Enter the policyName to which the requested certificate is mapped. **Example:** "FileBasedPolicy"
isKeyRequired *String*	(Mandatory) Enter if private key is necessary in the certificate package. **Example:** "false"

Name	Description
isChainRequired *String*	(Mandatory) Enter if certificate chain is necessary along with the code signing certificate. **Example:** "true"

Response Structure

- **Status Code:** 200 OK
- **Message:** Successfully downloaded the certificate ZIP file.
- **Headers:**
  - **Content-Type:** application/json
- **Response:** Certificate Info Package is downloaded.

Status Codes

HTTP Code	appStatusCode	Response Message
200 OK	-	null
400 Bad Request	CODE_SIGNING_00240	Policy does not exist.
400 Bad Request	VALIDATION_ERROR_0004	Invalid 'serialNumber'.
403 Forbidden	CODE_SIGNING_00218	Permissions are not there to download the certificate(s) for the requested Policy Name.
403 Forbidden	CODE_SIGNING_0032	Permissions are not there to sign the file/hash due to cert group/policy permissions are disabled.
500 Internal Server Error	CODE_SIGNING_0070	Signing Key is not mapped to the given policy.
500 Internal Server Error	CODE_SIGNING_0073	Certificate is not present in the cert inventory.
500 Internal Server Error	CODE_SIGNING_0087	Signing Key is Revoked/Expired.
500 Internal Server Error	CODE_SIGNING_0060	Error in generating the cert files during the Sign +/Certificate Package Creation.

HTTP Code	appStatusCode	Response Message
500 Internal Server Error	CODE_SIGNING_00221	Error in generating the private key file during the Certificate File Downloading.
500 Internal Server Error	CODE_SIGNING_00217	Download operation failed for the requested Certificate(s).

Sample Request/Response

Use Case

This API is used for retrieving certificate file(s).

Request URL

```
https://<IP/HostName/TenantName>:<GWPORT>/avxapi/code-signing-download-certificate?gwsource=external
```

Request Payload

```
{
  "payload": {
    "commonName": "AppViewXCertificate",
    "serialNumber": "18:C1:CD:90:72:FA:84:5A:87:30:7B:F7:11:47:69:B5:B0:BB:D5:57",
    "policyName": "FileBasedPolicy",
    "isKeyRequired": "false",
    "isChainRequired": "true"
  }
}
```

Sample Response

```
A ZIP file containing the full certificate chain will be downloaded.
```

Reference

Understanding the sample URL:

- **IP/HostName/TenantName**: Replace with the actual IP address, hostname, or tenant name based on the specific configuration in AppViewX.
  - **IP**: A unique identifier assigned to each device connected to a computer network that uses the Internet Protocol for communication

    The IP address will be included in the endpoint URL for an on-prem deployment.
  - **HostName**: A human-readable label assigned to a device (host) on a network

The hostname will be included in the endpoint URL for an on-prem deployment.

- **TenantName**: An identifier label for a tenant given to indicate which tenant's data the API request will access/modify

  The tenant name will be included in the endpoint URL for a SaaS deployment.

- **GWPORT**: AppViewX gateway port

  A gateway port refers to a network port through which data is sent and received to communicate with a gateway in an on-prem deployment.

  Example: **31443**

- **avxapi**: Path parameter value (static) that is part of the endpoint's URL
- **Endpoint**: Endpoint of the API, for example: **execute-hook**
- **gwsource**: Source or origin of a gateway, for example: **external**.

Code Signing Get Policy Key Data

The "code-signing-get-policy-key-data" API enables users to retrieve detailed information about TSA configuration, hashing algorithms, and mapped keys for specific code signing policies.

Before you begin

- Ensure that the certificate has been enrolled and mapped to the signing policy.

Request Structure

Endpoint:	`/code-signing-get-policy-key-data`
Type:	`GET`
Sample URL:	`https://<IP/HostName/TenantName>:<GWPORT>/avxapi/code-signing-get-policy-key-data?gwsource=external&filterType=hashBasedSigning` To understand the elements of the sample URL, click here.
Content-Type:	application/json

Input Parameter

Name	Description
Token *String* **(header)**	(Mandatory) Use token retrieved from login API. **Example:** eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJwbGF0Zm9ybSIsImF1ZCI6ImF2eCIsImNsaWVudElkIjoiOTcwNzRlNDEtOGFmOS00NTZkLTlhNjQtZjBjNGJiOTA4MDQ4IiwiaXNzIjoiYXZ4IiwiZXhwIjoxNjUwMzY5MzY3LCJncmFudCB0eXBlIjoiY2xpZW50X2NyZWRlbnRpYWxzIn0.HZnkuUEjXIeqJWqpqiNWFHqIDI7GYf4cWx 6VwbjGD_0
sessionId *String* **(header)**	(Mandatory) After successfully logging in, a unique identifier assigned to a user's session after successful authentication. The session ID remains valid until it expires. The session ID is a string value. **Example:** "ce7f1a14-2bf9-4e4a-89a8-bc780a255813"
username *String* **(header)**	(Mandatory) AppViewX login username, represented as a string value. **Example:** "User"
password *String* **(header)**	(Mandatory) AppViewX login username, represented as a string value. **Example:** "AppViewX@123"
gwsource *String* **(query)**	(Mandatory) Source from which the request is triggered. The values can be: • web • external **Type:** *String*
filterType *String* **(query)**	(Optional) Enter the criteria for filtering policies and key data to ensure precise and relevant data retrieval. • -- • hashBasedSigning • fileBasedSigning

Response Structure

- **Status Code:** 200 OK
- **Message:** null
- **Headers:**
  - **Content-Type:** application/json

Response Parameters

Name	Description
response	Contains the response attributes for the get policy request.
message	Success message or failure description in case of error.
appStatusCode	Application specific status code for the response. Will be non-null for failure response.
tags	More info in case of failure response.

Status Codes

HTTP Code	appStatusCode	Response Message
200 OK	-	null
400 Bad Request	VALIDATION_ERROR_0004	Input fields does not comply with the validation criteria. Please recheck the input payload::[Filter Type can only be fileBasedSigning/hashBasedSigning]
400 Bad Request	AVX_GW_001	Invalid Request. Please contact AppViewX Support.

Sample Request/Response

Use Case

This API is designed to retrieve detailed information about TSA configuration, hashing algorithms, and mapped keys for specific code signing policies.

Request URL

```
https://<IP/HostName/TenantName>:<GWPORT>/avxapi/code-signing-get-policy-key-data?gwsource=external&filterType=hashBasedSigning
```

Request Payload

```
NA
```

Sample Response for Hash Based Policy and Key Data


```
{
  "response": {
    "policyKeysData": {
      "HashBasedPolicy": {
        "signingKeys": [
          "AppViewXCertificate=0F:63:61:90:16:F5:5D:0B:BA:87:46:89:6C:F2:BC:4B"
        ],
        "signingHashAlgorithm": [
          "SHA-256"
        ],
        "timeStampingURL": [
          "http://timestamp.globalsign.com/tsa/r6advanced1"
        ]
      },
      "HashBasedSigning_Policy_Windows": {
        "signingKeys": [
          "CertCodeSigningEJBCA_UploadCert_RSA4096HSM.appviewx.com=37:9B:BE:A2:DE:81:E6:37:68:21:5B:BE:78:6C:F9:38:00:02:D6:21"
        ],
        "signingHashAlgorithm": [
          "SHA-256"
        ],
        "timeStampingURL": [
          "http://timestamp.digicert.com"
        ],
        "fileTypes": null
      }
    },
    "totalCount": 0
  }
}
```


Sample Response for File Based Policy and Key Data


```
{
  "response": {
    "policyKeysData": {
      "Signing_Policy_Upload_Cert_File_Jar": {
```

```
      "signingKeys": [
        "CertCodeSigningEJBCA_UploadCert_RSA4096HSM.appviewx.com=37:9B:BE:A2:DE:81:E6:37:68:21:5B:BE:78:6C:F9:38:00:02:D6:21"
      ],
      "signingHashAlgorithm": [
        "SHA-256"
      ],
      "timeStampingURL": [
        "http://timestamp.entrust.net/TSS/RFC3161sha2TS"
      ],
      "fileTypes": [
        "JAR"
      ]
    },
    "FileBasedPolicy": {
      "signingKeys": [
        "AppViewXCertificate=0F:63:61:90:16:F5:5D:0B:BA:87:46:89:6C:F2:BC:4B"
      ],
      "signingHashAlgorithm": [
        "SHA-256"
      ],
      "timeStampingURL": [
        "http://timestamp.digicert.com"
      ],
      "fileTypes": [
        "JAR",
        "PS1",
        "EXE",
        "JS"
      ]
    }
  },
  "totalCount": 0
 }
}
```


Sample Response for All Policies Key Data


```
{
 "response": {
```



```
"policyKeysData": {
  "HashBasedPolicy": {
    "signingKeys": [
      "AppViewXCertificate=0F:63:61:90:16:F5:5D:0B:BA:87:46:89:6C:F2:BC:4B"
    ],
    "signingHashAlgorithm": [
      "SHA-256"
    ],
    "timeStampingURL": [
      "http://timestamp.globalsign.com/tsa/r6advanced1"
    ]
  },
  "HashBasedSigning_Policy_Windows": {
    "signingKeys": [
      "CertCodeSigningEJBCA_UploadCert_RSA4096HSM.appviewx.com=37:9B:BE:A2:DE:81:E6:37:68:21:5B:BE:78:6C:F9:38:00:02:D6:21"
    ],
    "signingHashAlgorithm": [
      "SHA-256"
    ],
    "timeStampingURL": [
      "http://timestamp.digicert.com"
    ],
    "fileTypes": null
  },
  "Signing_Policy_Upload_Cert_File_Jar": {
    "signingKeys": [
      "CertCodeSigningEJBCA_UploadCert_RSA4096HSM.appviewx.com=37:9B:BE:A2:DE:81:E6:37:68:21:5B:BE:78:6C:F9:38:00:02:D6:21"
    ],
    "signingHashAlgorithm": [
      "SHA-256"
    ],
    "timeStampingURL": [
      "http://timestamp.entrust.net/TSS/RFC3161sha2TS"
    ],
    "fileTypes": [
      "JAR"
    ]
```

```
    },
    "FileBasedPolicy": {
      "signingKeys": [
        "AppViewXCertificate=0F:63:61:90:16:F5:5D:0B:BA:87:46:89:6C:F2:BC:4B"
      ],
      "signingHashAlgorithm": [
        "SHA-256"
      ],
      "timeStampingURL": [
        "http://timestamp.digicert.com"
      ],
      "fileTypes": [
        "JAR",
        "PS1",
        "EXE",
        "JS"
      ]
    }
  },
  "totalCount": 0
 }
}
```


What's Next

- **Code Signing Get Added Keys for Policy**
- **Code Signing Get Added Meta Info for Policy**.

Reference

Understanding the sample URL:

- **IP/HostName/TenantName**: Replace with the actual IP address, hostname, or tenant name based on the specific configuration in AppViewX.
  - **IP**: A unique identifier assigned to each device connected to a computer network that uses the Internet Protocol for communication

    The IP address will be included in the endpoint URL for an on-prem deployment.
  - **HostName**: A human-readable label assigned to a device (host) on a network

The hostname will be included in the endpoint URL for an on-prem deployment.

- **TenantName**: An identifier label for a tenant given to indicate which tenant's data the API request will access/modify

  The tenant name will be included in the endpoint URL for a SaaS deployment.

- **GWPORT**: AppViewX gateway port

  A gateway port refers to a network port through which data is sent and received to communicate with a gateway in an on-prem deployment.

  Example: **31443**

- **avxapi**: Path parameter value (static) that is part of the endpoint's URL
- **Endpoint**: Endpoint of the API, for example: **execute-hook**
- **gwsource**: Source or origin of a gateway, for example: **external**.

Code Signing Get Added Keys for Policy

The "Code Signing Get Added Keys for Policy" API allows users to retrieve the keys added to a specified code signing policy. It enables developers and administrators to access the list of cryptographic keys associated with a particular policy, facilitating seamless management and configuration of code signing processes.

Before you begin

- Configure the signing policy with relevant details, ensuring mapping to the enrolled certificate (also identified as the signing key on the signing policy page).
- Ensure that the requested keys are included in the policy.

Request Structure

Endpoint:	`/code-signing-get-added-keys-for-policy`
Type:	`POST`
Sample URL:	`https://<Tenant_name/Host_name>:<portno>/avxapi/code-signing-get-added-keys-for-policy?gwsource=external`
Content-Type:	application/json

Input Parameter

Name	Description
Token *String* **(header)**	(Mandatory) Use token retrieved from login API. **Example:** eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJwbGF0Zm9ybSIsImF1ZCI6ImF2eCIsImNsaWVudElkIjoiOTcwNzRlNDEtOGFmOS00NTZkLTlhNjQtZjBjNGJiOTA4MDQ4IiwiaXNzIjoiYXZ4IiwiZXhwIjoxNjUwMzY5MzY3LCJncmFudCB0eXBlIjoiY2xpZW50X2NyZWRlbnRpYWxzIn0.HZnkuUEjXIeqJWqpqiNWFHqIDI7GYf4cWx 6VwbjGD_0
sessionId *String* **(header)**	(Mandatory) After successfully logging in, a unique identifier assigned to a user's session after successful authentication. The session ID remains valid until it expires. The session ID is a string value. **Example:** "ce7f1a14-2bf9-4e4a-89a8-bc780a255813"
username *String* **(header)**	(Mandatory) AppViewX login username, represented as a string value. **Example:** "User"
password *String* **(header)**	(Mandatory) AppViewX login username, represented as a string value. **Example:** "AppViewX@123"
gwsource *String* **(query)**	(Mandatory) Source from which the request is triggered. The values can be: • web • external **Type:** *String*
policyName *String* **(Payload)**	(Optional) Policy Name is utilized to retrieve all keys associated with a specific signing policy. **Example:** "FileBasedPolicy"

Response Structure

- **Status Code:** 200 OK
- **Message:** null
- **Headers:**
  - **Content-Type:** application/json

Response Parameters

Name	Description
response	Contains the response attributes for the get added keys for policy request.
message	Success message or failure description in case of error.
appStatusCode	Application specific status code for the response. Will be non-null for failure response.
tags	More info in case of failure response.

Status Codes

HTTP Code	appStatusCode	Response Message
200 OK	-	null
400 Bad Request	VALIDATION_ERROR_0004	Invalid 'policyName'
400 Bad Request	CODE_SIGNING_00240	Policy does not exist
400 Bad Request	VALIDATION_ERROR_0004	'policyName' should have at least '5' characters
403 Forbidden	CODE_SIGNING_0035	Permissions are not there to get the policy signing keys data.
500 Internal Server Error	CODE_SIGNING_0009	Get Signing Keys Operation Failed for the given policy.

Sample Request/Response

Use Case

This API is intended to fetch details regarding all the mapped keys for a particular policy using policy name.

Request URL

```
https://<Tenant_name/Host_name>:<portno>/avxapi/code-signing-get-added-keys-for-policy?gwsource=external
```

Request Payload

```
{
  "payload": {
    "policyName": "FileBasedPolicy"
  }
}
```

Sample Response

```
{
  "response": {
    "data": [
      {
        "keyName": "AppViewXCertificate",
        "keyType": "RSA",
        "expirationDate": "12/10/2025",
        "caName": "AppViewX Intermediate CA",
        "keyId": "6757d579b1b426758a12abf9",
        "serialNumber": "0F:63:61:90:16:F5:5D:0B:BA:87:46:89:6C:F2:BC:4B",
        "defaultKey": false,
        "certificateHash": "73e72a34aa9b0903f1bc7996f5ac28050443a8df2bf3b0e56ea225ba7c30008c",
        "groupName": "Default"
      }
    ],
    "iTotalDisplayRecords": 1,
    "keyId": "6757d579b1b426758a12abfa",
    "totalCount": 0
  }
}
```

Reference

Understanding the sample URL:

- **IP/HostName/TenantName**: Replace with the actual IP address, hostname, or tenant name based on the specific configuration in AppViewX.
  - **IP**: A unique identifier assigned to each device connected to a computer network that uses the Internet Protocol for communication

    The IP address will be included in the endpoint URL for an on-prem deployment.
  - **HostName**: A human-readable label assigned to a device (host) on a network

    The hostname will be included in the endpoint URL for an on-prem deployment.
  - **TenantName**: An identifier label for a tenant given to indicate which tenant's data the API request will access/modify

    The tenant name will be included in the endpoint URL for a SaaS deployment.
- **GWPORT**: AppViewX gateway port

  A gateway port refers to a network port through which data is sent and received to communicate with a gateway in an on-prem deployment.

  Example: **31443**
- **avxapi**: Path parameter value (static) that is part of the endpoint's URL
- **Endpoint**: Endpoint of the API, for example: **execute-hook**
- **gwsource**: Source or origin of a gateway, for example: **external**.

Code Signing Get Added Meta Info for Policy

The "Code Signing Get Added Meta Info for Policy" API retrieves additional meta information associated with a specific code signing policy. This API enables users to access supplementary details, such as versioning information, build numbers, or any other custom fields added to the policy.

Before you begin

- Configure the signing policy with relevant details, ensuring mapping to the enrolled certificate (also identified as the signing key on the signing policy page).
- Ensure that the requested meta information are configured in the policy.

Request Structure

Endpoint:	`/code-signing-get-added-meta-info-for-policy`

Type:	POST
Sample URL:	https://<Tenant_name/Host_name>:<portno>/avxapi/code-signing-get-added-meta-info-for-policy?gwsource=external
Content-Type:	application/json

Input Parameter

Name	Description
Token *String* **(header)**	(Mandatory) Use token retrieved from login API. **Example:** eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJwbGF0Zm9ybSIsImF1ZCI6ImF2eCIsImNsaWVudElkIjoiOTcwNzRlNDEtOGFmOS00NTZkLTlhNjQtZjBjNGJiOTA4MDQ4IiwiaXNzIjoiYXZ4IiwiZXhwIjoxNjUwMzY5MzY3LCJncmFudCB0eXBlIjoiY2xpZW50X2NyZWRlbnRpYWxzIn0.HZnkuUEjXIeqJWqpqiNWFHqIDI7GYf4cWx 6VwbjGD_0
sessionId *String* **(header)**	(Mandatory) After successfully logging in, a unique identifier assigned to a user's session after successful authentication. The session ID remains valid until it expires. The session ID is a string value. **Example:** "ce7f1a14-2bf9-4e4a-89a8-bc780a255813"
username *String* **(header)**	(Mandatory) AppViewX login username, represented as a string value. **Example:** "User"
password *String* **(header)**	(Mandatory) AppViewX login username, represented as a string value. **Example:** "AppViewX@123"
gwsource *String* **(query)**	(Mandatory) Source from which the request is triggered. The values can be: • web • external **Type:** *String*

Input Parameter (continued)

Name	Description
policyName *String* **(Payload)**	(Optional) Policy Name is utilized to retrieve all keys associated with a specific signing policy. **Example:** "FileBasedPolicy"

Response Structure

- **Status Code:** 200 OK
- **Message:** null
- **Headers:**
  - **Content-Type:** application/json

Response Parameters

Name	Description
response	Contains the response attributes for the get added meta information for policy request.
message	Success message or failure description in case of error.
appStatusCode	Application specific status code for the response. Will be non-null for failure response.
tags	More info in case of failure response.

Status Codes

HTTP Code	appStatusCode	Response Message
200 OK	-	null
400 Bad Request	VALIDATION_ERROR_0004	Invalid 'policyName'
400 Bad Request	VALIDATION_ERROR_0004	'policyName' should have at least '5' characters
400 Bad Request	CODE_SIGNING_00240	Policy does not exist.

HTTP Code	appStatusCode	Response Message
403 Forbidden	CODE_SIGNING_0034	Permissions are not there to get the policy meta info data.
500 Internal Server Error	CODE_SIGNING_0010	Get Signing Meta Info Operation Failed for the given policy.

Sample Request/Response

Use Case

This API is intended to retrieve all the meta information mapped to a specific policy using the policy name.

Request URL

```
https://<Tenant_name/Host_name>:<portno>/avxapi/code-signing-get-added-meta-info-for-policy?gwsource=external
```

Request Payload

```
{
  "payload": {
    "policyName": "FileBasedPolicy"
  }
}
```

Sample Response

```
{
  "response": {
    "data": [
      {
        "metaName": "Build",
        "type": "text",
        "isMandatory": "Yes",
        "metaInfoId": "6756d1bfdf58417d9f5eb3b0"
      },
      {
        "metaName": "Integer",
        "type": "integer",
        "isMandatory": "No",
        "metaInfoId": "6756d1c7df58417d9f5eb3b1"
      }
    ],
    "iTotalDisplayRecords": 2,
```

```
  "metaInfoId": "6756d157df58417d9f5eb3ac",
  "totalCount": 0
 }
}
```


Reference

Understanding the sample URL:

- **IP/HostName/TenantName**: Replace with the actual IP address, hostname, or tenant name based on the specific configuration in AppViewX.
  - **IP**: A unique identifier assigned to each device connected to a computer network that uses the Internet Protocol for communication

    The IP address will be included in the endpoint URL for an on-prem deployment.
  - **HostName**: A human-readable label assigned to a device (host) on a network

    The hostname will be included in the endpoint URL for an on-prem deployment.
  - **TenantName**: An identifier label for a tenant given to indicate which tenant's data the API request will access/modify

    The tenant name will be included in the endpoint URL for a SaaS deployment.
- **GWPORT**: AppViewX gateway port

  A gateway port refers to a network port through which data is sent and received to communicate with a gateway in an on-prem deployment.

  Example: **31443**
- **avxapi**: Path parameter value (static) that is part of the endpoint's URL
- **Endpoint**: Endpoint of the API, for example: **execute-hook**
- **gwsource**: Source or origin of a gateway, for example: **external**.