Garantir

GaraSign Windows User Guide

Version 1.28.1

Table of Contents

Preface

Document Information

Title	GaraSign Windows User Guide
Product Version	1.28.1
Release Date	March 2025

Trademarks

Disclaimer

Garantir makes no representations or warranties with respect to the contents of this document and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Furthermore, Garantir reserves the right to revise this publication and to make changes from time to time in the content hereof without the obligation upon Garantir to notify any person or organization of any such revisions or changes.

We have attempted to make these documents complete, accurate, and useful, but we cannot guarantee them to be without error or otherwise perfect. When we discover errors or omissions, or they are brought to our attention, we endeavor to correct them in succeeding releases of the product.

Please send any constructive comments on the contents of this document to the following email address: support@garantir.io

Overview

This document details how to install, configure, operate, troubleshoot, and uninstall the GaraSign Key Storage Provider (KSP) for Microsoft Windows. This KSP allows applications that integrate with the appropriate Cryptography Next Generation (CNG) APIs to sign and decrypt data using GaraSign. While it is possible to use this KSP programmatically, this documentation only discusses integrating with the Microsoft tools listed in the Usage section. For more information on programmatic access, please consult Microsoft's CNG API documentation.

Intended Audience

All products produced by Garantir are designed to be installed, configured, operated, and maintained by personnel with the necessary knowledge, skill, training, and qualifications to safely perform their duties. This document is intended for personnel responsible for installing, configuring, operating, and/or troubleshooting signing clients on Windows systems. It is assumed that the readers of this document and the users of its content are proficient with:

- The Microsoft Windows operating system
- Working from the command line

- Security concepts including, but not limited to, authentication, authorization, digital signatures, and logging
- The signing tools they use

Document Conventions

This document uses the following conventions to easily distinguish commands and alert you to important information.

Commands

Commands to be executed from a command prompt are provided in italics with a gray background, as shown below:

```
exampleCommand.exe /switch1
```

In some cases, commands will need to be edited with customer-specific values. These parts of the commands will be highlighted and in brackets, as shown below:

```
exampleCommand.exe /switch1 <insert some value>
```

Warnings

To help reduce the chance of data loss or corruption, this document provides warnings inside a red box with a warning icon, as shown below:

⚠ **Warning**: Always exercise caution when performing security-related duties.

Components

This KSP is made up of the following components:

1. Installer – Automates the installation and uninstallation of the KSP.
    a. GarantirKSP-x64.msi (for 64-bit platforms)
    b. GarantirKSP-x86.msi (for 32-bit platforms)
2. Configuration File (config.ini) – Used to configure authentication, remote server information, etc.
3. KSP DLL (GRSKSP.dll) – Provides functionality to integrate with GaraSign via CNG APIs.
4. Certificate Loading Utility (CertificateLoader.exe) – Loads the user's certificates into the Local User's Personal Windows Certificate Store. This utility is scheduled to run every time a user logs in but can also be run manually.
5. PKCS11 DLL (grsp11.dll) – Provides functionality to integrate with GaraSign via PKCS#11.
6. GaraSign Wrapper Tool (garasign.exe) – Wrapper tool to assist with signing, certificate retrieval, and listing of keys.

Supported Mechanisms

Signature

This KSP supports signing data using RSA or ECDSA (P_256, P_384, P_521) with the following hash algorithms:

1. MD5 (Supported for legacy systems. Not recommended unless required.)

2. SHA-1 (Supported for legacy systems. Not recommended unless required.)
3. SHA-256
4. SHA-384
5. SHA-512

Decryption
This KSP supports decrypting data using the following decryption schemes:

1. RSA-OAEP with SHA-1 and MGF1
2. RSA-OAEP with SHA-256 and MGF1
3. RSA-OAEP with SHA-384 and MGF1
4. RSA-OAEP with SHA-512 and MGF1
5. RSA with PKCS#1 v1.5 padding

Key Agreement
This KSP supports the following key agreement schemes:

1. ECDH_P256
2. ECDH_P384
3. ECDH_P521

Authentication
This KSP supports the following authentication methods:

1. Kerberos (i.e., Windows Authentication)
2. Client-Certificate (i.e., Mutual TLS)
3. Username and Password
4. OpenID Connect (OIDC)

Installation

Planning Your Installation
There are three ways to install this KSP; choose the method that best suits your needs:

1. **GUI Installer**

   Intended for end users who are installing this KSP for their own use and prefer a graphical user interface to the command line.

2. **Silent Installer**

   Intended for administrators of a network of computers that wish to "push" an installation of this KSP on multiple computers without requiring user interaction.

3. **Automated Installation & Configuration**

   Intended for users who have been given a client-bundle (typically named client.zip) that contains the installer package and user credentials.

Prerequisites

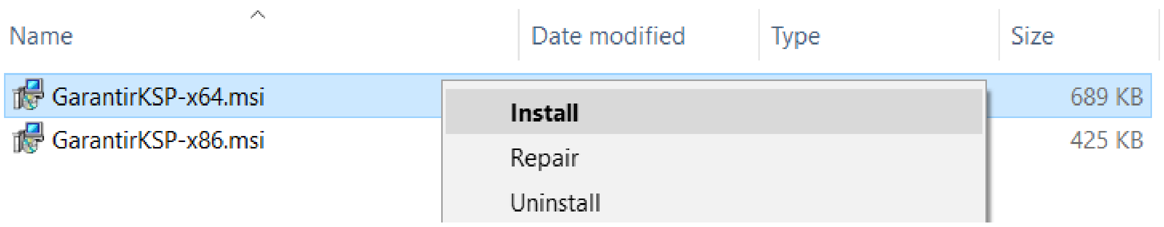
The following items are needed to install this KSP:

1. A computer running one of the following versions of Microsoft Windows:
   a. Windows 10
   b. Windows 11
   c. Windows Server 2019
   d. Windows Server 2022
2. Administrator access to the target computer
3. One of the following installers:
   a. GarantirKSP-x64.msi (for 64-bit platforms)
   b. GarantirKSP-x86.msi (for 32-bit platforms)
4. (Optional) The configuration file you wish to set as the default for all users on this system
5. (Optional) A pre-bundled installer package, typically named client.zip, that contains the items listed above

GUI Installer

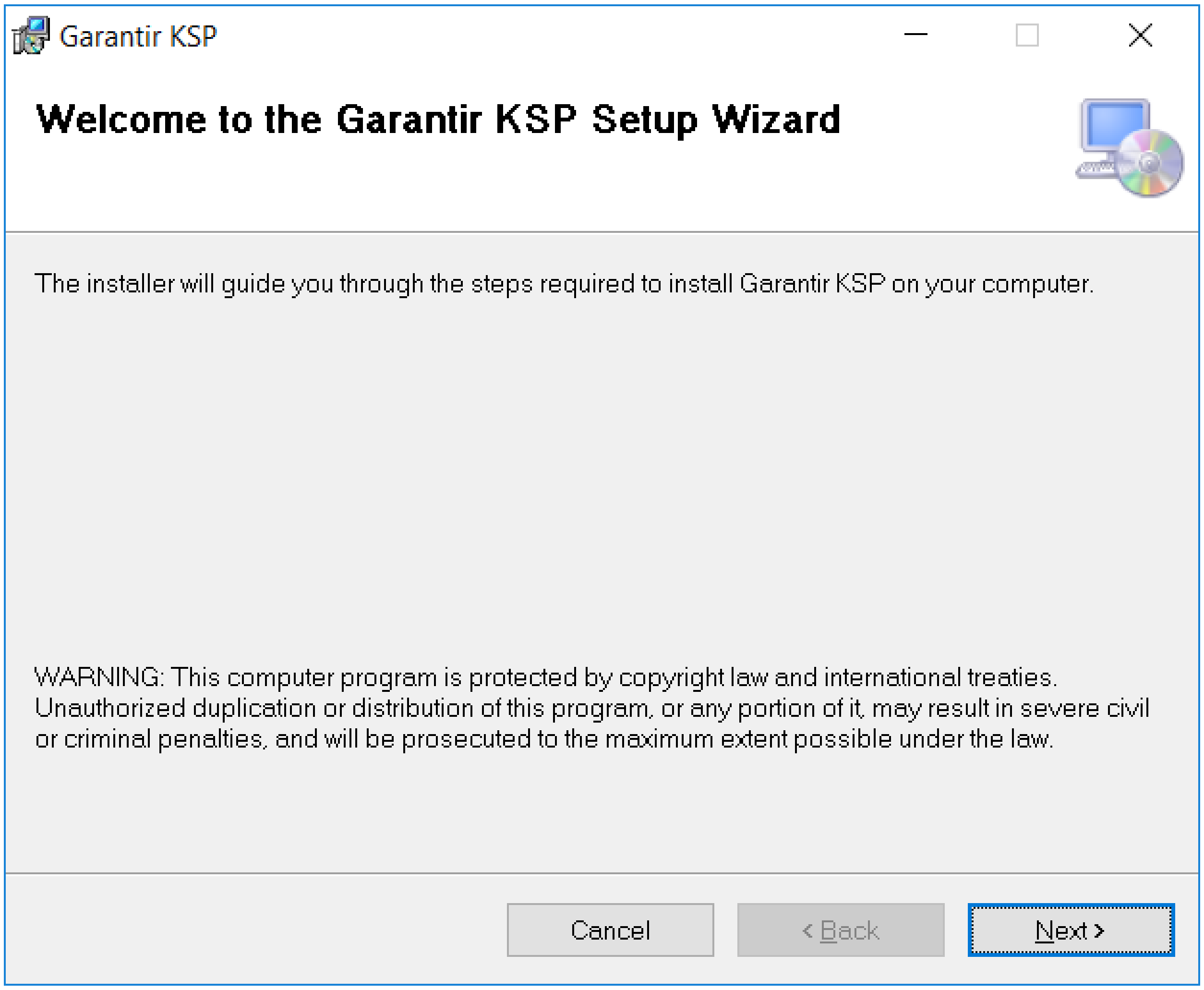
To install the KSP via the GUI, execute the following steps:

1. Right click the appropriate installer for your platform and click Install.



Name	Date modified	Type	Size
GarantirKSP-x64.msi			689 KB
GarantirKSP-x86.msi			425 KB

2. At the Welcome dialog click Next.

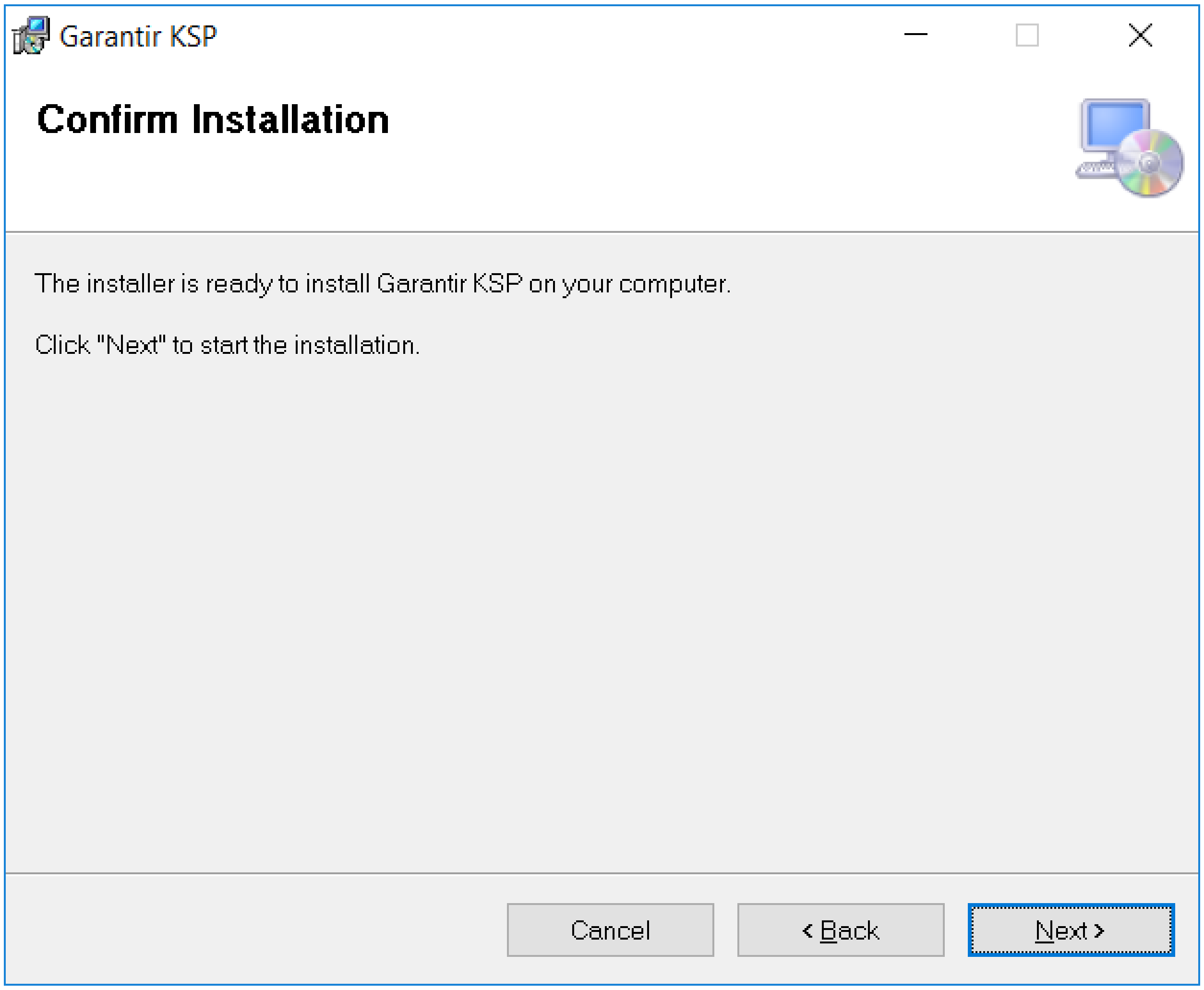


Welcome to the Garantir KSP Setup Wizard

The installer will guide you through the steps required to install Garantir KSP on your computer.

WARNING: This computer program is protected by copyright law and international treaties. Unauthorized duplication or distribution of this program, or any portion of it, may result in severe civil or criminal penalties, and will be prosecuted to the maximum extent possible under the law.

3. At the Confirmation dialog click Next.



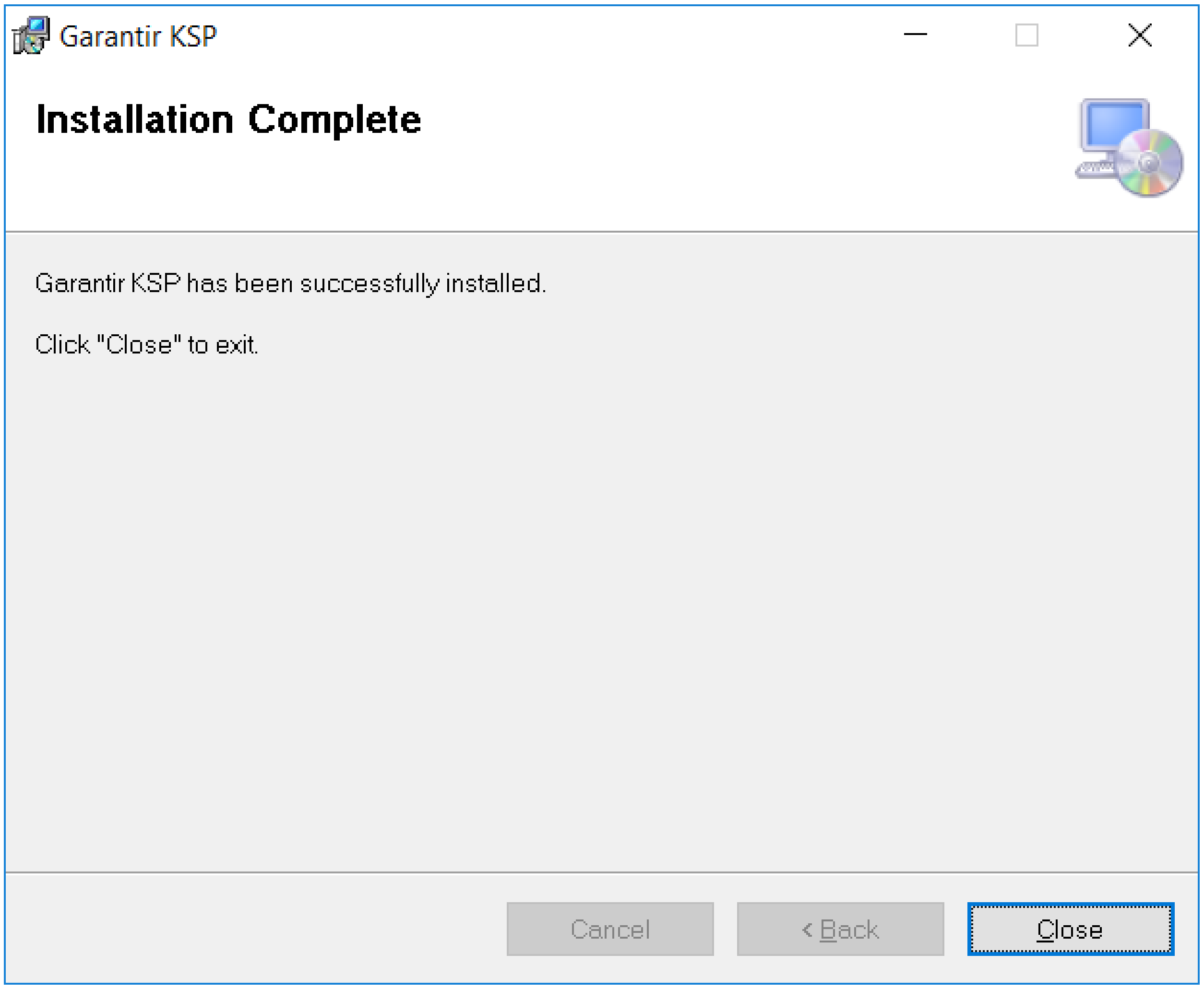
Garantir KSP

Confirm Installation

The installer is ready to install Garantir KSP on your computer.

Click "Next" to start the installation.

4. Click Yes to allow the app to make changes to your device.
5. Click Close on the Installation Complete dialog.



Installation Complete

Garantir KSP has been successfully installed.

Click "Close" to exit.

Silent Installer

To silently install the KSP, perform the following steps:

1. Open a command prompt as an Administrator.
2. Execute the following command, depending on your configuration (note: commands need to be altered with the correct file path(s) for your computer):
   32-bit Platform without Default Configuration File
   msiexec /i "<path/to/installers/GarantirKSP-x86.msi>" /quiet /qn

   32-bit Platform with Default Configuration File
   msiexec /i "<path/to/installers/GarantirKSP-x86.msi>" CONFIGFILE="<path/to/default/config.ini>" /quiet /qn

   64-bit Platform without Default Configuration File
   msiexec /i "<path/to/installers/GarantirKSP-x64.msi>" /quiet /qn

   64-bit Platform with Default Configuration File

msiexec /i "<path/to/installers/GarantirKSP-x64.msi>" CONFIGFILE="<path/to/default/config.ini>" /quiet /qn

3. (Optional) Execute the following command and verify that the application is found:
   where CertificateLoader

   If the result of the command displays the full path to CertificateLoader.exe then the installation completed successfully.

Automated Installation and Configuration

To automate the installation and configuration, perform the following steps:

1. Unzip the bundled installation package (client.zip)
2. Open a PowerShell command prompt as an Administrator.
3. Navigate to the extracted folder
4. Execute the following command:
   .\setup.ps1

Configuration

Configuration of the KSP is done by editing the INI configuration file, located in

%USERPROFILE%/AppData/Roaming/Garantir/GRS/config.ini

After a successful installation of the KSP the default config.ini will be located in

%ALLUSERSAPPDATA%/Garantir/GRS/config.ini

Then, once the newly installed CertificateLoader has run for a given user (scheduled to run whenever a user logs in), this file will be copied over to

%USERPROFILE%/AppData/Roaming/Garantir/GRS/config.ini

Editing the default config.ini (i.e., the one in *%ALLUSERSAPPDATA%/Garantir/GRS/config.ini)* will result in changing the configuration for all users on that computer (once they run CertificateLoader again). Editing the user's config.ini (i.e., the one in *%USERPROFILE%/AppData/Roaming/Garantir/GRS/config.ini)* will result in changing the configuration for that user only.

Configuration Values

Property Name	Section	Property Value
Name	Server	[String] The DNS name or IP address of GaraSign
Port	Server	[int] The port number that the GaraSign listens on
URL	Server	[String] The path in the URL of the GaraSign
AlwaysUseGlobal	Config	[int] Set to 1 to use the global config.ini file instead of the user's local file. Set to 0 to use the user's config.ini file. Default = 0.
Enable	Cache	[int] Set to 1 to cache the session token. Set to 0 to not cache the session token. Default = 1.

TTL	Cache	[int] The time, in seconds, that the cache value is considered valid. Default = 300.
Enable	UI	[int] Set to 1 to have a UI dialog displayed for each signature operation. Set to 0 to not have any UI dialogs displayed. Default = 0.
Enable	Log	[int] Set to 1 to have log files generated. Set to 0 to not have log files generated. Default = 0.
Folder	Log	[String] The directory to write log files to.
Type	OpenID	[String] Set to either AZURE, OKTA, or GOOGLE. Default = AZURE.
OfflineMode	OpenID	[int] Set to 1 to request a refresh token and use it as long as it is valid. Set to 0 to prompt have the user prompted every time authentication occurs. Default = 0.
IssuerURL	OpenID	[String] The Issuer URL of the OpenID Connect provider.
GRSClientAppID	OpenID	[String] The Client Application ID.
FullScope	OpenID	[String] In most cases this should be set to openid. For customers on the legacy approach, this should be set to api://<GRSWebAPIAppID>/<scope>, where <scope> is the actual scope of the GRS REST Web API App (e.g access_as_user).
GRSClientAppSecret	OpenID	[String] The Client Application secret. This is required when using Client Credentials Flow with a Client Secret (Azure), and Authorization Code Flow with PKCE and Client Secret (Google), otherwise it must be empty
GRSClientAppCertificateHash	OpenID	[String] The fingerprint (in hexadecimal format) of the Client Certificate to use. This is required when using Client Credentials Flow with a Client Certificate, otherwise it must be empty.
GRSClientAppCertificateKeyPin	OpenID	[String] The PIN of the private key for the client certificate to use. This is required when using Client Credentials Flow with a Client Certificate and the corresponding key lives in a Smart Card, otherwise it must be empty.
Timeout	InProgress	[int] The number of seconds to wait during IN_PROGRESS transactions before timing out. Default = 120 seconds (i.e., 2 minutes).
Count	Users	[int] For future functionality. Leave as 1 for now.
Authentication	User1	[String] The authentication method to use. Options are *Kerberos* and *Normal*. Use *Kerberos* to enable Windows Authentication (Kerberos via SPNEGO). Use *Normal* to enable certificate authentication (i.e., mutual TLS) or username and password authentication.
AllowGlobalKerberos	User1	[int] Set to 0 to restrict Kerberos authentication to intranet sites. Set to 1 to enable Kerberos authentication to all sites, including those on the internet. Default = 0.
Username	User1	[String] The username to authenticate with if authenticating with username and password.

Password	User1	[String] The password to authenticate with if authenticating with username and password.
CertificateHash	User1	[String] The thumbprint of the certificate from the *My* (or *Personal*) Windows Certificate Store to use when authenticating via client certificate.

Authentication

This KSP supports Kerberos, Username & Password, Client Certificate, and OpenID Connect authentication.

Kerberos

To enable Kerberos authentication, set the *authentication* value to *Kerberos.* You can optionally choose to restrict Kerberos authentication to intranet sites or open it up to non-intranet sites by setting the *AllowGlobalKerberos* value. By default, Kerberos authentication is restricted to intranet sites only.

Kerberos Example (Restricted to Intranet Sites)

```
[user1]

Authentication=Kerberos

AllowGlobalKerberos=0
```

Kerberos Example (Open to Internet)

```
[user1]

Authentication=Kerberos

AllowGlobalKerberos=1
```

Username & Password

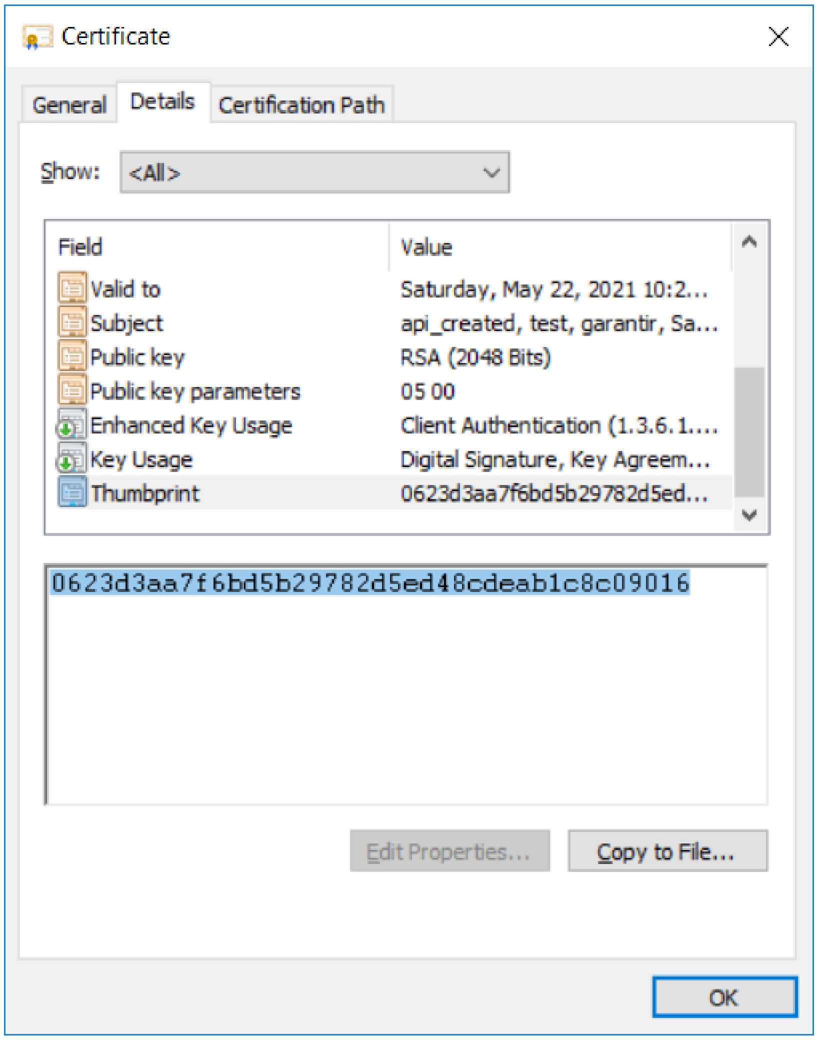
To enable username & password authentication, set the *authentication* value to *Normal* and set the *username* and *password* values to the ones provided by your GaraSign system administrator in the *user1* section of the KSP's configuration file. Note: the username & password values are **not** the same as your Windows login credentials so do **not** place those values in this configuration file.

Username & Password Example

```
[User1]

Authentication=Normal

Username=signing_client_username

Password=$ign_Ev3ry+h1nG!
```

Client Certificate

To enable client certificate authentication, import your client certificate into the *Personal* store of the Local User's Windows Certificate store and retrieve its Thumbprint value from the Details tab of the Windows Certificate UI (see Figure below). Then, in the KSP's configuration file (config.ini), set the *authentication* value to *Normal* and set the *certificateHash* value to the Thumbprint value retrieved.



Field	Value
Valid to	Saturday, May 22, 2021 10:2...
Subject	api_created, test, garantir, Sa...
Public key	RSA (2048 Bits)
Public key parameters	05 00
Enhanced Key Usage	Client Authentication (1.3.6.1....
Key Usage	Digital Signature, Key Agreem...
Thumbprint	0623d3aa7f6bd5b29782d5ed...

```
0623d3aa7f6bd5b29782d5ed48cdeab1c8c09016
```

Note: the method to obtain a client certificate is customer-specific and out of scope for this document. Some organizations may choose to centrally generate the key material and certificates and distribute them back to the end user in a password protected file format (e.g., .pfx, .p12, etc.). Others may require the client to generate their own key material, send the Certificate Signing Request (CSR) to the Certificate Authority / Registration Authority, and import the signed certificate response. Please consult your GaraSign system administrator for instructions on how your organization handles client certificates.

Client Certificate Example

```
[user1]

authentication=Normal

certificateHash=0623d3aa7f6bd5b29782d5ed48cdeab1c8c09016
```

Logging

To enable logging, set the *Enable* property to *1* and the *Folder* property to the desired directory in the *Log* section of the KSP's configuration file.

Note 1: Logging degrades performance, so it is recommended to only enable it to troubleshoot issues.

Note 2: The configured folder for the log files must exist and be writable by the process that will utilize the KSP.

Logging Example

```
[Log]

Enable=1

Folder=C:\Users\username\logs
```

Cache

To increase performance, this KSP supports caching the session token so that each call to sign or decrypt does not need to perform its own authentication, but rather will share the session token from the first valid authentication. To enable session token caching, set the *Enable* property to *1* and the *TTL* value to the desired time, in seconds, that the session token should be cached for in the *Cache* section of the KSP's configuration file.

Note 1: The maximum time that the session token can be valid for is 7 hours and 59 minutes = 28740 seconds.

Note 2: The session token is protected on disk using the Windows-provided DPAPI with additional entropy added for further security.

⚠ **Warning**: While steps are taken to protect the cache, it should still be considered sensitive information. Users should take appropriate steps to safeguard their cache from unauthorized access.

Cache Example

```
[Cache]

Enable=1

TTL=28740
```

Clearing the Cache

At times it may be necessary to clear the cache. There are two options to do this – manually and via the TTL value. To manually clear the cache, navigate to *%USERPROFILE%/AppData/Roaming/Garantir/GRS* and delete the *Cache* folder. To clear the cache via the TTL value, set the TTL to something very small (e.g., 1) and rerun *CertificateLoader.* This will cause the old cache to be considered expired and a new cache file to be created. This latter approach resets the cache whereas the first approach just clears it.

Certificate Loading

Before signing can occur, the certificates for the user account must be loaded into the user's Personal Certificate Store. Once the certificates are loaded, data can be signed by specifying the desired certificate from the user's Personal Certificate Store.

Loading Certificates

To load the available certificates for your account, perform the following steps:

1. Open a command prompt as a regular user (i.e., **not** as an Administrator)
2. Execute the following command: *CertificateLoader*

Upon successful completion, a message will be displayed stating "X certificates were loaded to the certificate store" where X will be replaced by the number of certificates available to the user.

Unloading Certificates

At times it may be necessary to remove previously loaded certificates. While this can be done manually via the Windows-provided *certmgr* utility, *CertificateLoader* provides a more automated approach. To unload the previously loaded certificates for your account, perform the following steps:

1. Open a command prompt as a regular user (i.e., **not** as an Administrator)
2. Execute the following command: *CertificateLoader /unload*

Upon successful completion, a message will be displayed stating "X certificates were removed from the certificate store" where X will be replaced by the number of certificates that were removed.

Note: This action only removes the certificates from the local computer. It does **not** remove any certificate or private key material server-side.

Code Signing

There are many ways to sign code on Windows. The following sections provide information on how to integrate GaraSign with various signing tools.

GaraSign CLI

The GaraSign CLI (garasign.exe) is a utility for listing keys, exporting certificates, and signing. While the tool can be used for signing, it is recommended that it only be used to display the signing command, rather than perform actual signing.

To list keys with the GaraSign CLI, perform the following steps:

1. Open a command prompt as a regular user (i.e., **not** as an Administrator)
2. Execute the following command: *garasign listkeys*

To export a certificate with the GaraSign CLI, perform the following steps:

1. Open a command prompt as a regular user (i.e., **not** as an Administrator)
2. Execute the following command: *garasign export --key <key name> --outputDirectory <output directory>*

To sign a file with the GaraSign CLI, perform the following steps:

1. Open a command prompt as a regular user (i.e., **not** as an Administrator)
2. Execute the following command: *garasign sign --key <key name> --inputFile <file to sign>*

⚠ **Warning**: This signing command will sign the file in place and perform signature verification before signing. See the help menu for more options when signing.

Signtool

To use the KSP with signtool.exe, specify the thumbprint of the certificate you wish to use in your signtool command via the */sha1* switch. For example, the following command signs the file *to_sign.exe* using the private key corresponding to the specified certificate and SHA256 as the underlying hash algorithm:

signtool sign /fd SHA256 /sha1 "<thumbprint of your signing certificate>" to_sign.exe

Note 1: The *sha1* switch is only used to identify which certificate to use. The signature is **not** being generated using SHA-1.

Note 2: The KSP only manages the signing of the data and does not interfere with the other signtool switches or functionality. You can use other switches in addition to the ones displayed in the example command above. For example, one could additionally timestamp the binaries using an external TSA via the following command:

signtool sign /fd SHA256 /sha1 "<thumbprint of your signing certificate>" /t "http://<url_and_path_to_your_TSA>" to_sign.exe

Please consult Microsoft's documentation for further information on using signtool.

PowerShell & VBScript Signing

The simplest way to sign a PowerShell or VBScript file with this KSP is to use *Signtool*. See the Code Signing

There are many ways to sign code on Windows. The following sections provide information on how to integrate GaraSign with various signing tools.

GaraSign CLI

The GaraSign CLI (garasign.exe) is a utility for listing keys, exporting certificates, and signing. While the tool can be used for signing, it is recommended that it only be used to display the signing command, rather than perform actual signing.

To list keys with the GaraSign CLI, perform the following steps:

3. Open a command prompt as a regular user (i.e., **not** as an Administrator)
4. Execute the following command: *garasign listkeys*

To export a certificate with the GaraSign CLI, perform the following steps:

3. Open a command prompt as a regular user (i.e., **not** as an Administrator)
4. Execute the following command: *garasign export --key <key name> --outputDirectory <output directory>*

To sign a file with the GaraSign CLI, perform the following steps:

3. Open a command prompt as a regular user (i.e., **not** as an Administrator)
4. Execute the following command: *garasign sign --key <key name> --inputFile <file to sign>*

Signtool section for instructions.

Nuget

To use the KSP with nuget.exe, specify the thumbprint of the certificate you wish to use in your nuget command via the --*certificate-fingerprint* switch. For example, the following command signs the file *to_sign.pkg* using the private key corresponding to the specified certificate and SHA256 as the hash algorithm:

```
nuget sign to_sign.pkg --hash-algorithm SHA256 --certificate-fingerprint "<thumbprint of your signing certificate>"
```

Java

GaraSign can integrate with Java-based tools and custom code through multiple means. On supported Linux platforms, the recommended approach is to use GaraSign's PKCS#11 provider (please see the GaraSign JCA/JCE User Guide for details on a pure Java integration). To integrate with PKCS#11, create a configuration file (e.g., grspkcs11.conf) and place the following content in it:

```
name = GaraSign

library = c:/windows/system32/grsp11.dll
```

Then pass the configuration file to the SunPKCS11 provider and load the keys through the JCE's KeyStore.

JarSigner

Pass the PKCS#11 configuration file to JarSigner while specifying the SunPKCS11 provider. For example:

```
jarsigner -keystore NONE -storetype PKCS11 -providerClass sun.security.pkcs11.SunPKCS11 -providerArg "/path/to/grspkcs11.conf" -storepass ignored -sigalg SHA256withRSA <file to sign> <key name>
```

Note: Timestamping can be added via the -tsa switch in JarSigner. Garantir strongly recommends that customers timestamp their code so that signatures remain valid past the expiration of the signing certificate.

Programmatic Use

To use the SunPKCS#11 integrated with GaraSign provider programmatically, perform the following steps:

1. Load the provider (not necessary if the provider was installed statically)
2. Load the provider's key store
3. Retrieve the desired key from the key store
4. Use the provider's signature implementation along with the retrieve key to sign

Example – Dynamic Provider instantiation:

```java
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.InputStream;
import java.io.PrintStream;
import java.lang.reflect.Constructor;
import java.lang.reflect.Method;
import java.nio.charset.StandardCharsets;
import java.security.KeyStore;
import java.security.PrivateKey;
import java.security.Provider;
import java.security.Security;
import java.security.Signature;
```

```
public class Main {

  public static void main(String[] args) throws Exception {
         // load provider
      ByteArrayOutputStream byteStream = new ByteArrayOutputStream();
      PrintStream ps = new PrintStream(byteStream);
      String providerName = "SunPKCS11-GRS";
      ps.println("name = " + providerName);
      String libPath = sanitizePath("c:/windows/system32/grsp11.dll");
      ps.println("library = \"" + libPath + "\"");
      Provider pkcs11Provider = getProvider(byteStream.toByteArray());
      Security.addProvider(pkcs11Provider);

      // load GaraSign PKCS11 keystore
      KeyStore ks = KeyStore.getInstance("PKCS11");
      ks.load(null, null);

      // retrieve desired private key
      String keyName = args[0];
      PrivateKey privKey = (PrivateKey)ks.getKey(keyName, null);

      // sign data with key
      String algorithm = args[1]; // examples: SHA256withRSA, SHA256withECDSA, etc.
      Signature sig = Signature.getInstance(algorithm, pkcs11Provider);
      sig.initSign(privKey);
      sig.update("hello, world".getBytes(StandardCharsets.UTF_8));
      byte[] signature = sig.sign();
  }

  private static Provider getProvider(byte[] configurationInfo) throws Exception {
    Provider PROV;
    String pkcs11Config = new String(configurationInfo, "UTF-8");

    try {
            // newer methods of Java use this approach
      Method configure = Provider.class.getMethod("configure", new Class[] {
String.class });

      PROV = Security.getProvider("SunPKCS11");
      PROV = (Provider)configure.invoke(PROV, new Object[] { pkcs11Config });

    } catch (NoSuchMethodException e) {
            // fall back for older methods of Java
      Constructor<?> SunPKCS11 =
Class.forName("sun.security.pkcs11.SunPKCS11").getConstructor(new Class[] {
InputStream.class });
      ByteArrayInputStream confStream = new
ByteArrayInputStream(pkcs11Config.getBytes());
      PROV = (Provider)SunPKCS11.newInstance(new Object[] { confStream });
    }

    return PROV;
  }

  private static String sanitizePath(String path) {
      if (path == null) {
        return null;
      }

      return path.replace("\\", "\\\\");
  }

}
```

Container Image Signing

GaraSign supports multiple methods of signing container images. On Windows, the recommended approach is to use cosign (from the Sigstore project) integrated with GaraSign's PKCS#11 provider. To do this, first list the tokens via the following command:

```
cosign pkcs11-tool list-tokens --module-path "c:\Windows\System32\grsp11.dll"
```

Retrieve the slot number (should just be 1) and then execute the following command:

```
cosign pkcs11-tool list-keys-uris --module-path "c:\Windows\System32\grsp11.dll" --slot-id 1 --pin 1234"
```

Retrieve the URI for the desired key and then execute the following command:

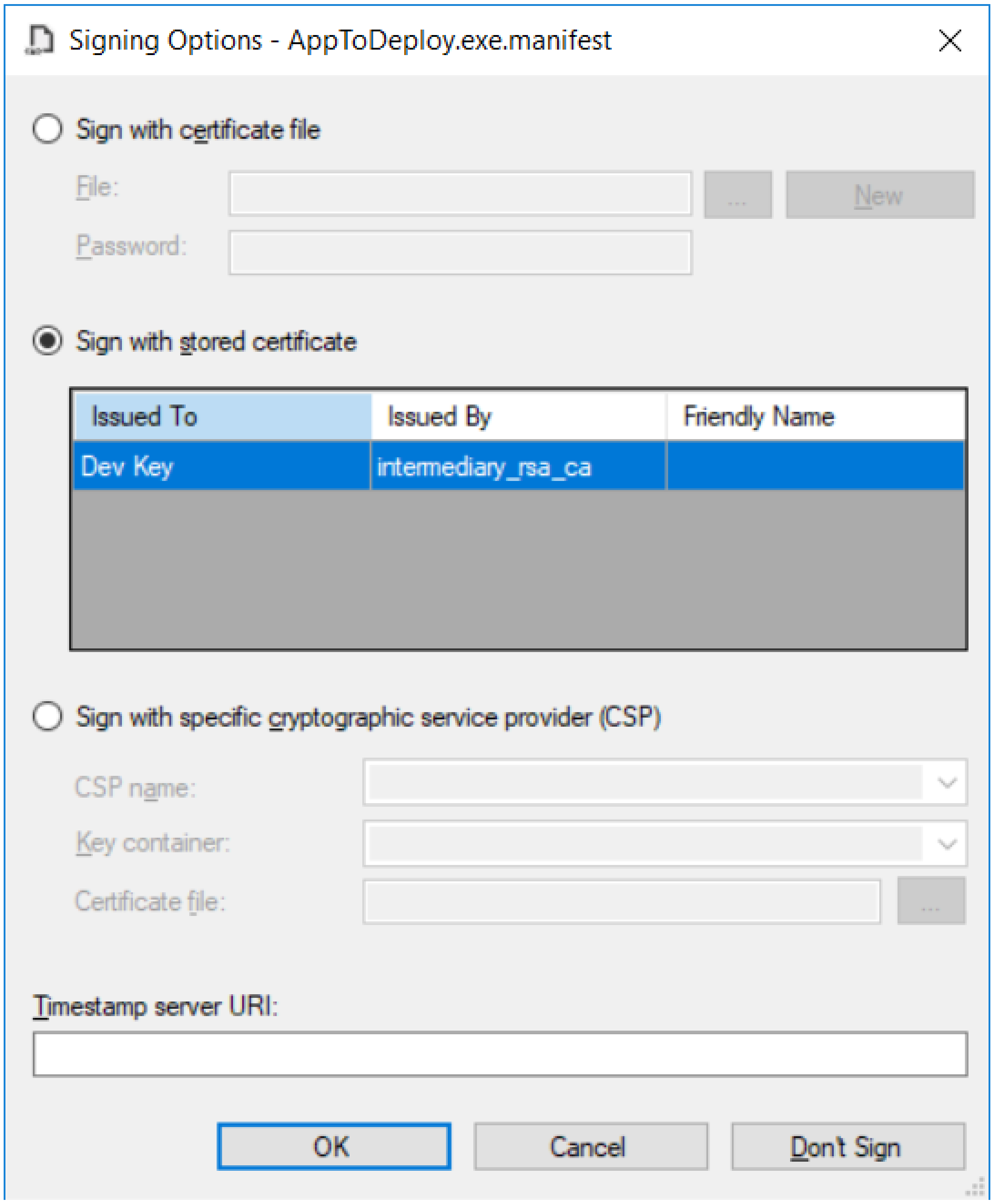
```
cosign sign --key "<PKCS11 URI>" <image to sign>
```

Mage & MageUI

To use the KSP with Mage.exe, specify the thumbprint of the certificate you wish to use in your Mage command via the *-CertHash* switch. For example, the following command signs the manifest MyApp.exe.manifest using the private key corresponding to the specified certificate:

```
mage -sign MyApp.exe.manifest -certHash "<thumbprint of your signing certificate>"
```

To use the KSP with MageUI.exe, select the *Sign with stored certificate* radio button and then select the desired stored certificate from the Signing Options dialog.


Signing Options - AppToDeploy.exe.manifest

○ Sign with certificate file

File:

Password:

● Sign with stored certificate

Issued To	Issued By	Friendly Name
Dev Key	intermediary_rsa_ca	

○ Sign with specific cryptographic service provider (CSP)

CSP name:

Key container:

Certificate file:

Timestamp server URI:

Click the OK button to initiate the signing process.

Strong Name

Sn.exe does not make use of certificates and so its usage is slightly different than the other command line tools.

Setup

Before using this KSP (technically, in this case, this CSP) with sn.exe, sn.exe must be configured and the public portion of the signing key must be extracted. To do this, perform the following steps:

1. Open a command prompt as Administrator
2. Execute the following command: *sn.exe -c "Garantir Remote Signing CSP" 24*

⚠ **Warning**: This command will set the *Garantir Remote Signing CSP* as the default CSP for the *entire* computer. If sn.exe is used with another CSP on this computer, sn.exe may need to be reconfigured to use that other CSP when that CSP is required.

3. Close the Administrator command prompt
4. Open a new command prompt as your regular user (i.e., **not** as an Administrator)
5. Execute the following command: *CertificateLoader /list*
6. From the printed list, choose the name of the key you would like to use and note its name (value displayed after the index number, **not** including the algorithm and key size in parentheses). For example, "sign_key@user".
   Note: The selected key **must** be an RSA key.
7. Execute the following command: *sn.exe -pc <value from step 6> publicKey.snk*
   Note: if you wish to use SHA256, use the following command instead: *sn.exe -pc <value from step 6> publicKey.snk sha256*

Note: if you require more than one signing key in your environment, repeat steps 6 and 7 as appropriate but be careful to rename the public key file (*publicKey.snk* in the commands above) for each signing key.

Signing

Once sn.exe has been configured and your public key has been extracted, you can begin strong name signing your assemblies. To do this, perform the following steps:

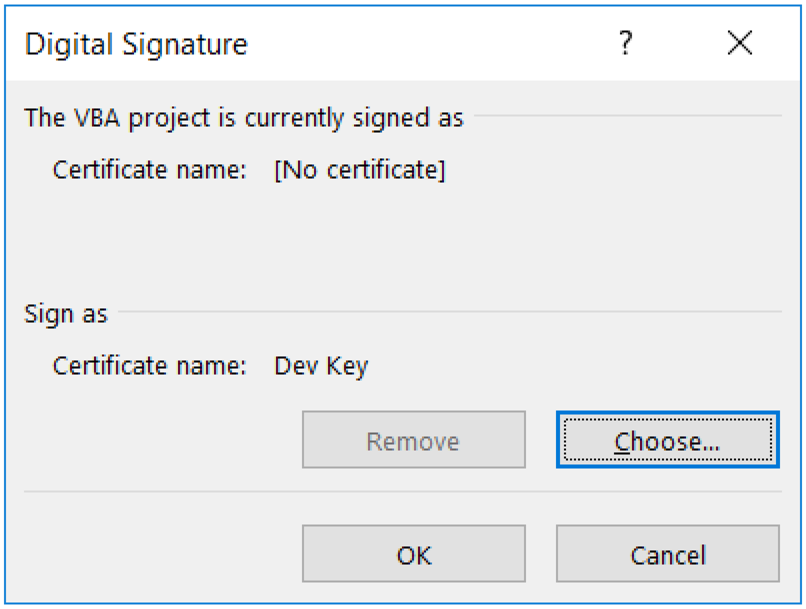
1. Open a Visual Studio Developer Command Prompt
2. Navigate to the folder containing the source code to compile
3. Execute the following command: *csc /delaySign+ /keyfile:<path to publicKey.snk> <source file>*
4. Close the Developer Command Prompt
5. Open a new command prompt as your regular user (i.e., **not** as an Administrator)
6. Execute the following command: *sn.exe -Rc <assembly result from step 3> <value from step 6 of setup>*

The assembly is now signed with a strong name. To validate the signature, execute the following command: *sn.exe -v <assembly>*

VBA/Macro Signing

To sign a Macro with this KSP, specify the appropriate loaded certificate from the Digital Signature dialog by executing the following steps:

1. From the *Tools* menu click on the *Digital Signature…* button to bring up the *Digital Signature* dialog.


Digital Signature

The VBA project is currently signed as

Certificate name: [No certificate]

Sign as

Certificate name: Dev Key

2. Click on the *Choose…* button, select the desired certificate, and then click OK.
   Note: It may be necessary to click the *More choices* link to find the desired certificate.
3. Click the OK button on the *Digital Signature* popup dialog.
4. Save the Macro to sign it.
5. Verify the Macro is signed by opening the *Digital Signature* dialog (*Tools->Digital Signature…*) again and viewing the *Certificate name* displayed at the top.

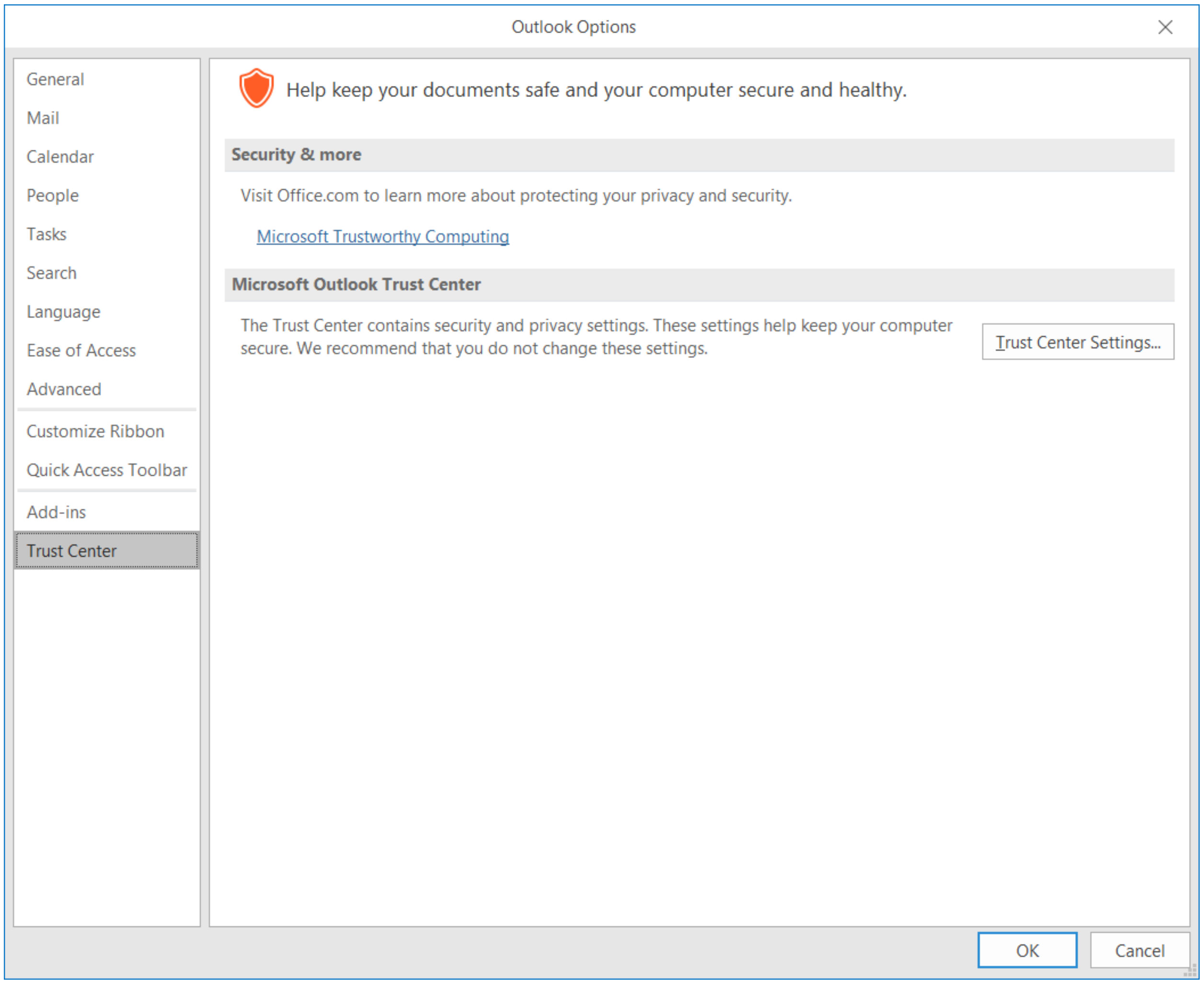
Note: At the time of writing this document, VBA macros are signed using MD5 as the underlying hash algorithm. This is forced by the VBA application, **not** by this KSP.

S/MIME

Outlook

Outlook can use this KSP to perform S/MIME signing and encryption/decryption. To configure the certificates to use for S/MIME, perform the following steps:

1. From the *File* menu click on *Options* and then click *Trust Center* on the left-hand panel.



Help keep your documents safe and your computer secure and healthy.

Security & more

Visit Office.com to learn more about protecting your privacy and security.

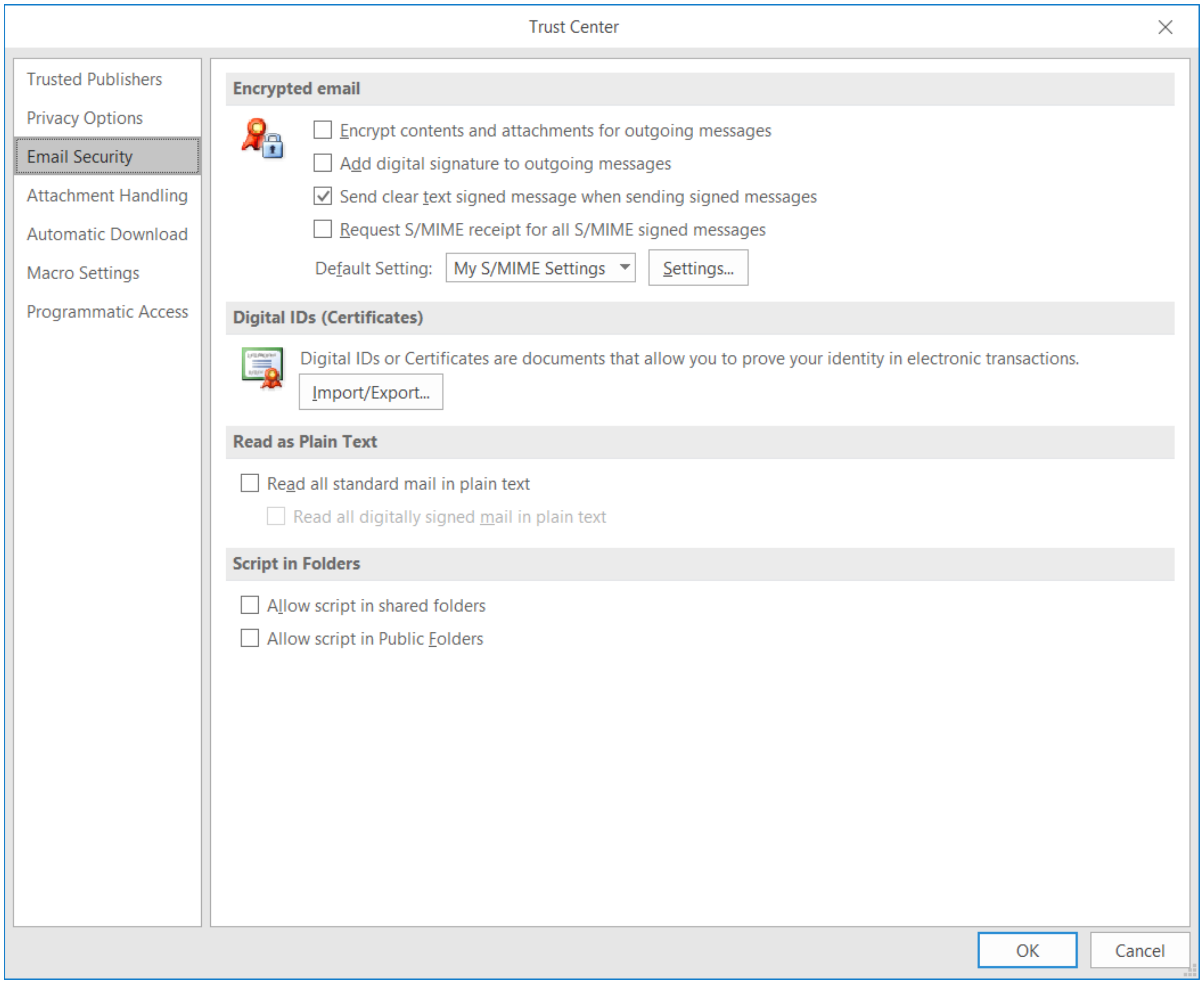
Microsoft Trustworthy Computing

Microsoft Outlook Trust Center

The Trust Center contains security and privacy settings. These settings help keep your computer secure. We recommend that you do not change these settings.

Trust Center Settings...

2. Under *Microsoft Outlook Trust Center* click *Trust Center Settings* to bring up the *Trust Center* popup and then click on *Email Security* on the left-hand panel.


Trust Center

Encrypted email

- [ ] Encrypt contents and attachments for outgoing messages
- [ ] Add digital signature to outgoing messages
- [x] Send clear text signed message when sending signed messages
- [ ] Request S/MIME receipt for all S/MIME signed messages

Default Setting: My S/MIME Settings

Digital IDs (Certificates)

Digital IDs or Certificates are documents that allow you to prove your identity in electronic transactions.

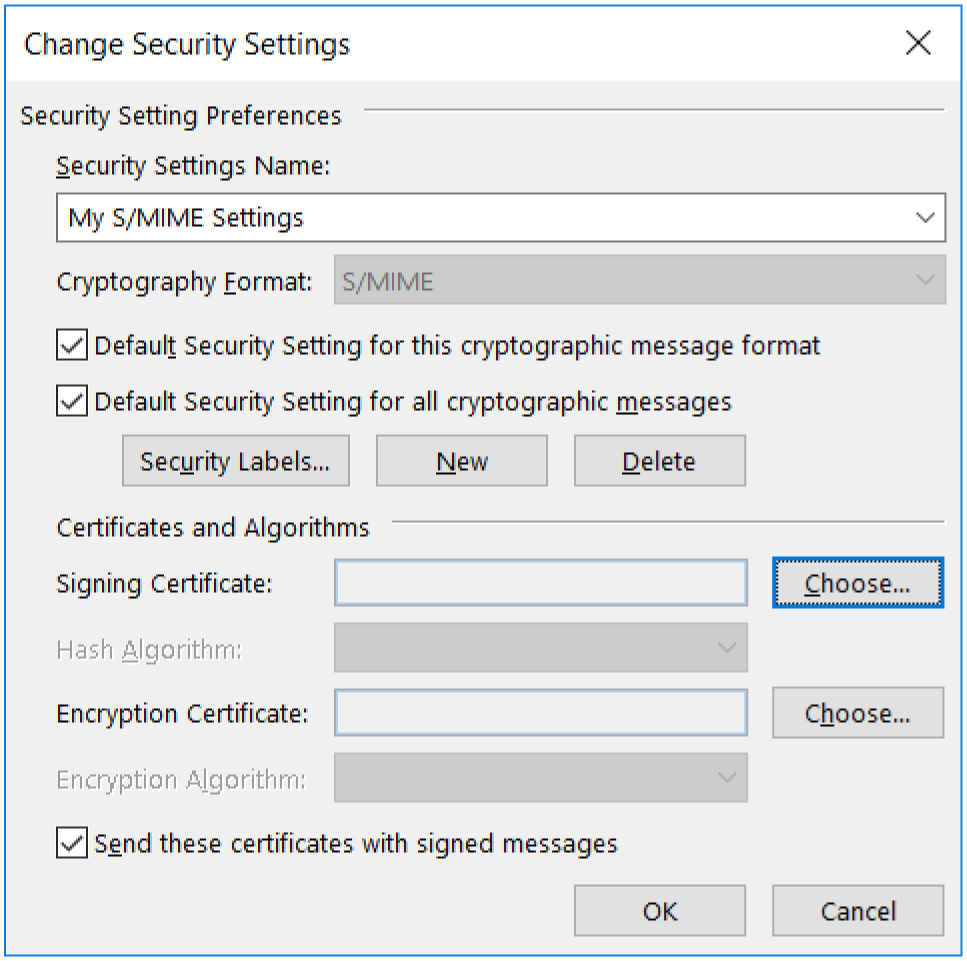
Read as Plain Text

- [ ] Read all standard mail in plain text
- [ ] Read all digitally signed mail in plain text

Script in Folders

- [ ] Allow script in shared folders
- [ ] Allow script in Public Folders

3. Under *Encrypted Email* click the *Settings…* button to bring up the *Change Security Settings* dialog.


Change Security Settings

Security Setting Preferences

Security Settings Name: My S/MIME Settings

Cryptography Format: S/MIME

☑ Default Security Setting for this cryptographic message format

☑ Default Security Setting for all cryptographic messages

Certificates and Algorithms

Signing Certificate:

Hash Algorithm:

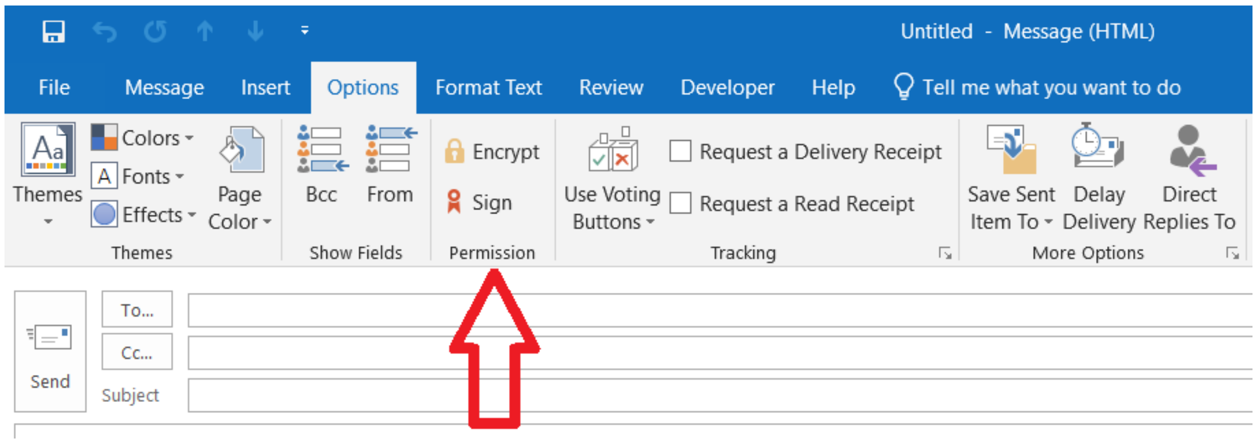
Encryption Certificate:

Encryption Algorithm:

☑ Send these certificates with signed messages

4. Click the *Choose…* button next to the desired certificate input (i.e., Signing Certificate, Encryption Certificate, or both) to select the appropriate certificate.
   Note: It may be necessary to click the *More choices* link to find the desired certificate.
5. Click OK on all the open dialogs to save the changes.

Outlook is now configured to use the KSP and selected certificates for S/MIME functionality. To use these features, click the appropriate button under the *Options* menu of your *New Email* dialog as shown below:



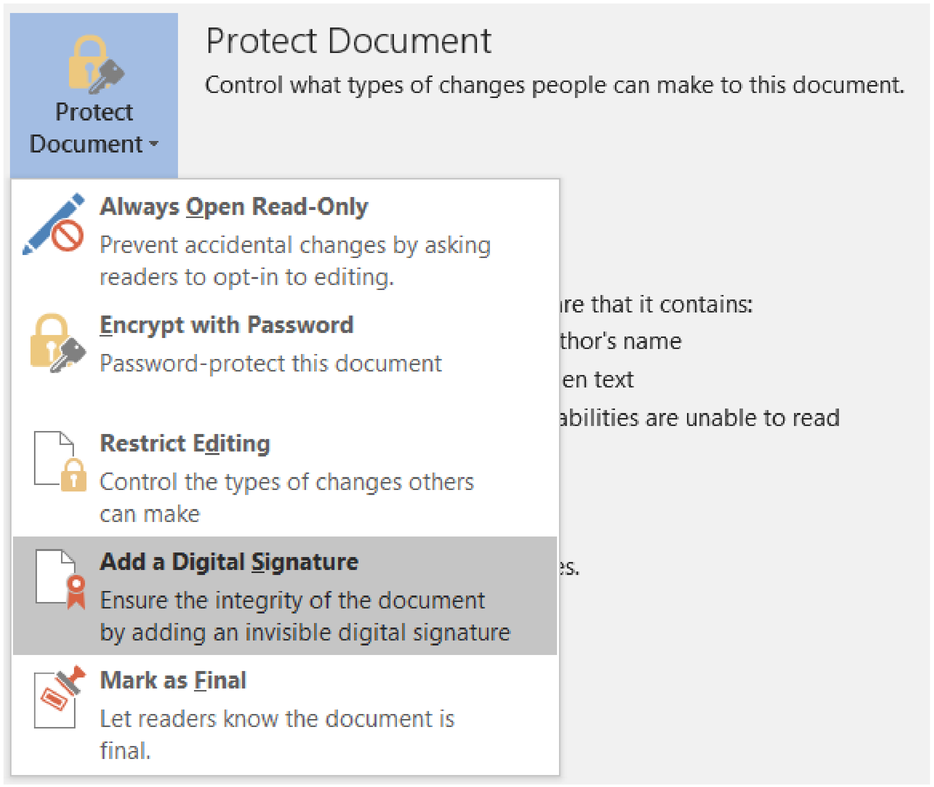
Document Signing

GaraSign can be used to sign many times of documents including PDFs, Microsoft Office documents, and more.

Microsoft Office Document Signing

The various Microsoft Office applications support digital signatures through similar means. While the exact button names slightly differ, the overall process is the same and is provided below:

1. From the *Info* section of the *File* menu click on *Protect Document* and select *Add a Digital Signature*



Protect Document
Control what types of changes people can make to this document.

Always Open Read-Only
Prevent accidental changes by asking readers to opt-in to editing.

Encrypt with Password
Password-protect this document

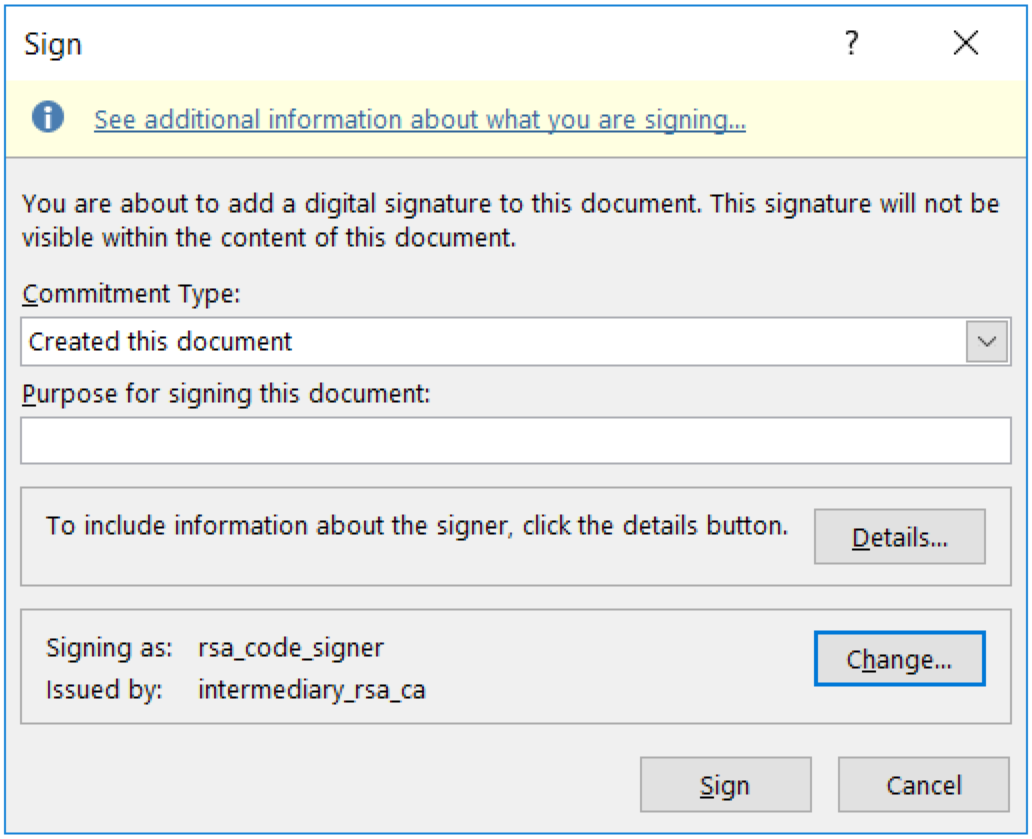
Restrict Editing
Control the types of changes others can make

Add a Digital Signature
Ensure the integrity of the document by adding an invisible digital signature

Mark as Final
Let readers know the document is final.

Note: Different Microsoft Office applications may have a different name for the dropdown. For example, Excel's dropdown is labeled *Protect Workbook* instead of *Protect Document.*

2. On the *Sign* dialog, select the appropriate *Commitment Type* and then select the desired certificate by clicking the *Change…* button.



Sign

See additional information about what you are signing...

You are about to add a digital signature to this document. This signature will not be visible within the content of this document.

Commitment Type:

Created this document

Purpose for signing this document:

To include information about the signer, click the details button.

Signing as: rsa_code_signer

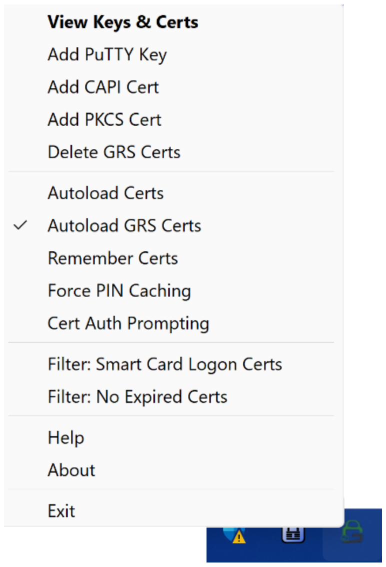
Issued by: intermediary_rsa_ca

Note: It may be necessary to click the *More choices* link to find the desired certificate.

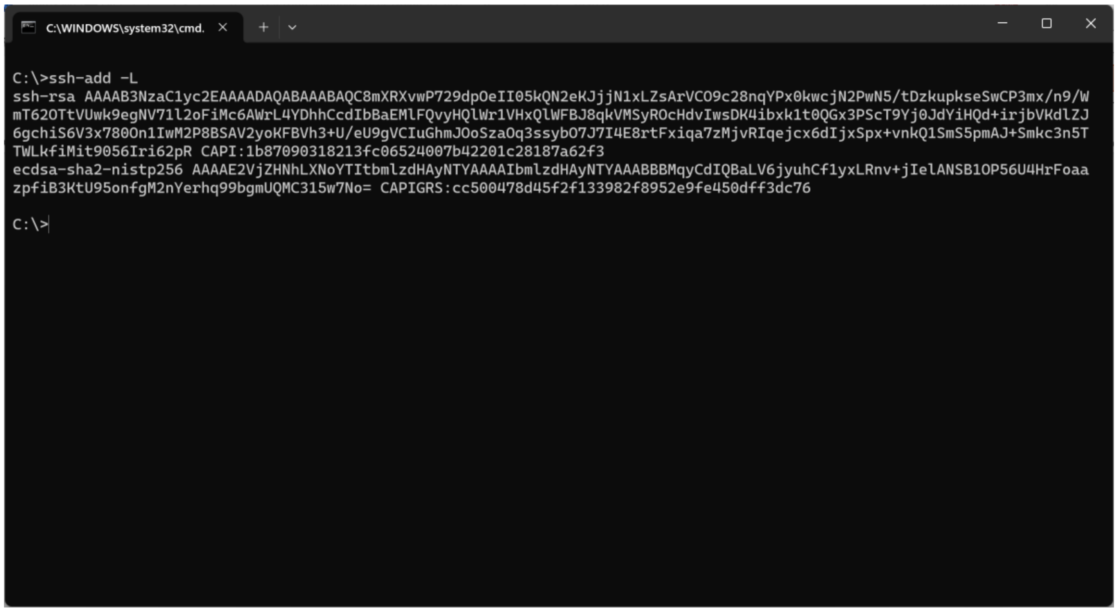
3. Click the *Sign* button to sign the document.

Secure Shell (SSH), Secure Copy Protocol (SCP), & Secure File Transfer Protocol (SFTP)

GaraSign supports SSH, SCP, and SFTP via ssh-agent and pageant. The management of the keys is shown via the Pageant agent in the system tray, as shown below.



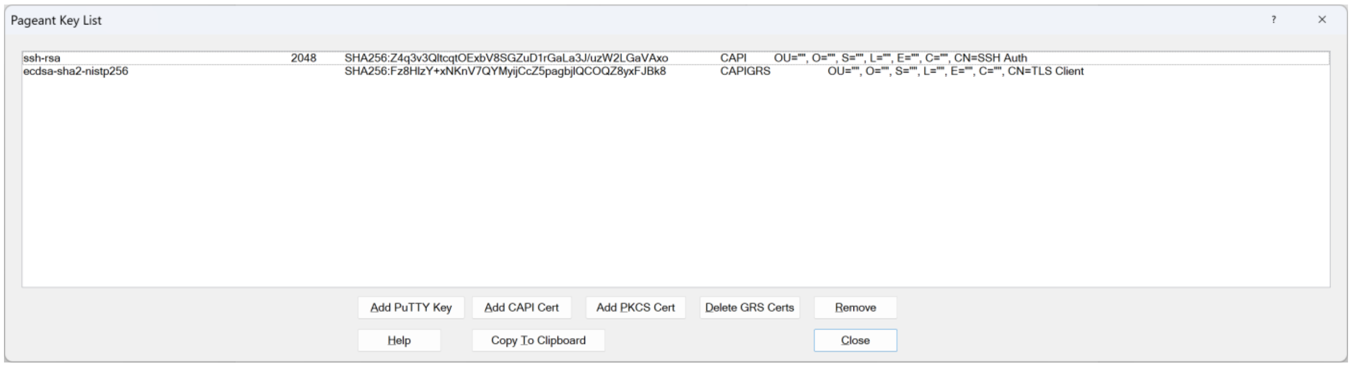
Keys can also be viewed and managed from the command line via the ssh-add command, as shown below.



```
C:\>ssh-add -L
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQC8mXRXvwP729dpOeII05kQN2eKJjjN1xLZsArVCO9c28nqYPx0kwcjN2PwN5/tDzkupkseSwCP3mx/n9/W
mT62OTtVUwk9egNV71l2oFiMc6AWrL4YDhhCcdIbBaEMlFQvyHQlWr1VHxQlWFBJ8qkVMSyROcHdvIwsDK4ibxk1t0QGx3PScT9Yj0JdYiHQd+irjbVKdlZJ
6gchiS6V3x780On1IwM2P8BSAV2yoKFBVh3+U/eU9gVCIuGhmJOoSzaOq3ssybO7J7I4E8rtFxiqa7zMjvRIqejcx6dIjxSpx+vnkQ1SmS5pmAJ+Smkc3n5T
TWLkfiMit9056Iri62pR CAPI:1b87090318213fc06524007b42201c28187a62f3
ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBMqyCdIQBaLV6jyuhCf1yxLRnv+jIelANSB1OP56U4HrFoaa
zpfiB3KtU95onfgM2nYerhq99bgmUQMC315w7No= CAPIGRS:cc500478d45f2f133982f8952e9fe450dff3dc76

C:\>
```

By default, keys with active certificates that have the Client Authentication extended key usage attribute will be populated for use. If additional GaraSign keys need to be loaded, click on the *Add CAPI Cert* button on the Pageant Key List popup. If non-GaraSign keys need to be loaded, click on the appropriate Add Key button on the Pageant Key List popup.

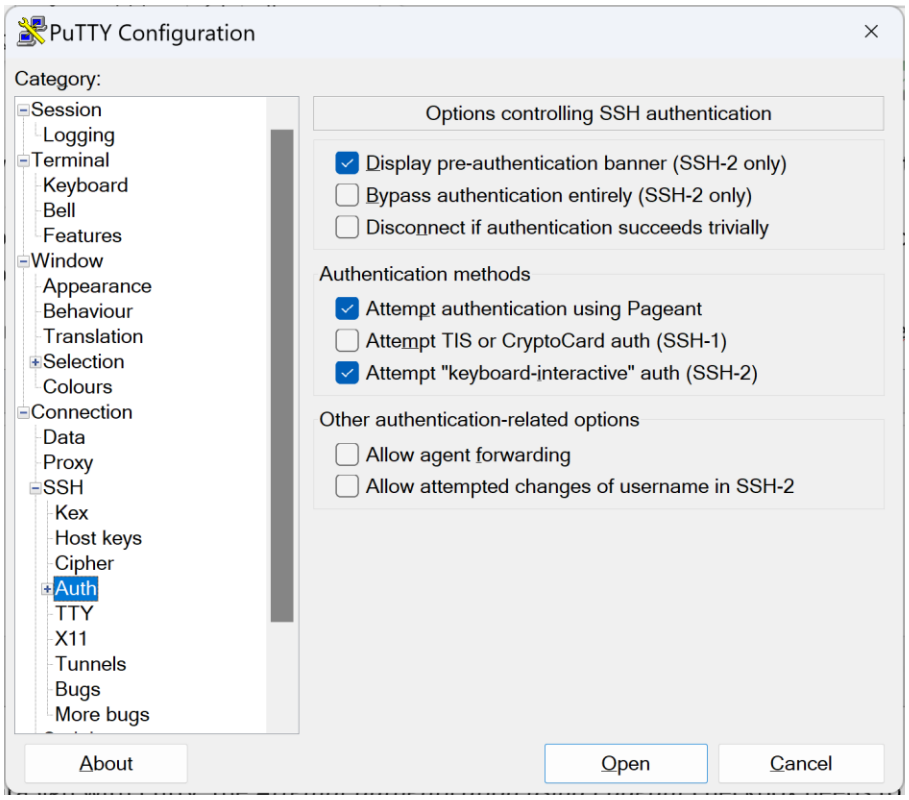
Note: keys and certificates are automatically populated in the Pageant Key List anytime *CertificateLoader* is run.



```
ssh-rsa              2048   SHA256:Z4q3v3QltcqtOExbV8SGZuD1rGaLa3J/uzW2LGaVAxo   CAPI     OU="", O="", S="", L="", E="", C="", CN=SSH Auth
ecdsa-sha2-nistp256         SHA256:Fz8HlzY+xNKnV7QYMyijCcZ5pagbjlQCOQZ8yxFJBk8   CAPIGRS          OU="", O="", S="", L="", E="", C="", CN=TLS Client
```

PuTTY

To use GaraSign with PuTTY, the *Attempt authentication using Pageant* checkbox needs to be checked. This feature is enabled by default so there should not be any additional configuration required for most users. Once this checkbox is checked, proceed to use PuTTY as you normally would.


PuTTY Configuration

Category:

Options controlling SSH authentication

- [✓] Display pre-authentication banner (SSH-2 only)
- [ ] Bypass authentication entirely (SSH-2 only)
- [ ] Disconnect if authentication succeeds trivially

Authentication methods

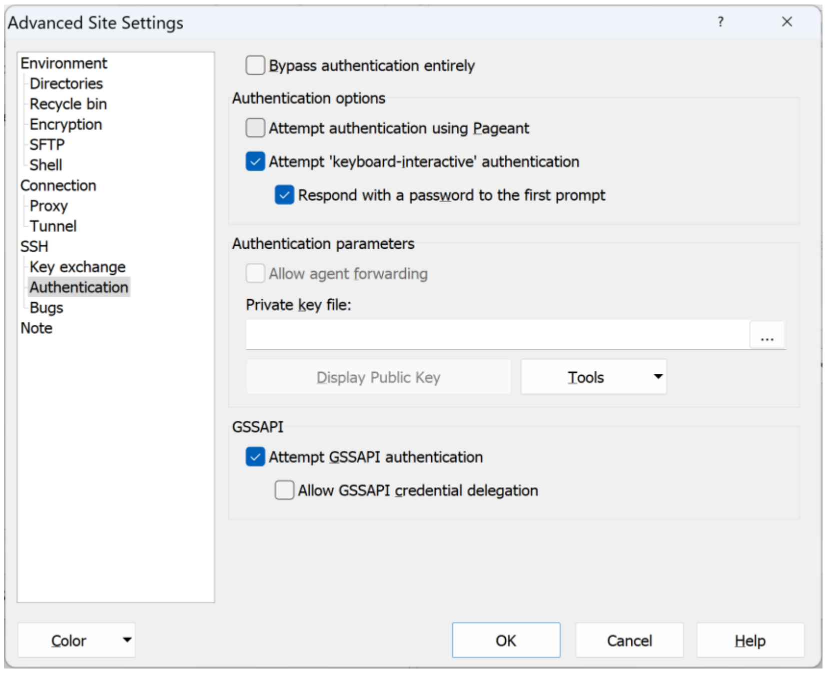
- [✓] Attempt authentication using Pageant
- [ ] Attempt TIS or CryptoCard auth (SSH-1)
- [✓] Attempt "keyboard-interactive" auth (SSH-2)

Other authentication-related options

- [ ] Allow agent forwarding
- [ ] Allow attempted changes of username in SSH-2

WinSCP

To use GaraSign with WinSCP, the *Attempt authentication using Pageant* checkbox needs to be checked. This feature is enabled by default so there should not be additional configuration required for most users. Once this checkbox is checked, proceed to use WinSCP as you normally would.


Advanced Site Settings

- [ ] Bypass authentication entirely

Authentication options

- [ ] Attempt authentication using Pageant
- [✓] Attempt 'keyboard-interactive' authentication
  - [✓] Respond with a password to the first prompt

Authentication parameters

- [ ] Allow agent forwarding

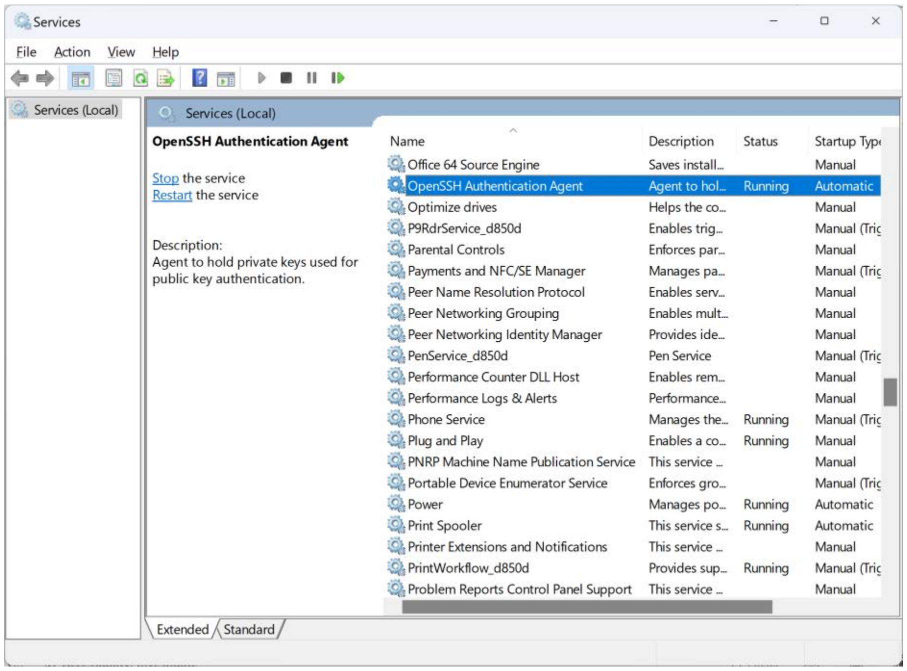
Private key file:

GSSAPI

- [✓] Attempt GSSAPI authentication
  - [ ] Allow GSSAPI credential delegation

OpenSSH

To use GaraSign with OpenSSH, the *OpenSSH Authentication Agent* service must be running. In many configurations this is enabled by default so users might not need to perform any additional configuration. Once the service is running, use OpenSSH as you normally would (e.g., *ssh username@host*).



OpenSSH Authentication Agent

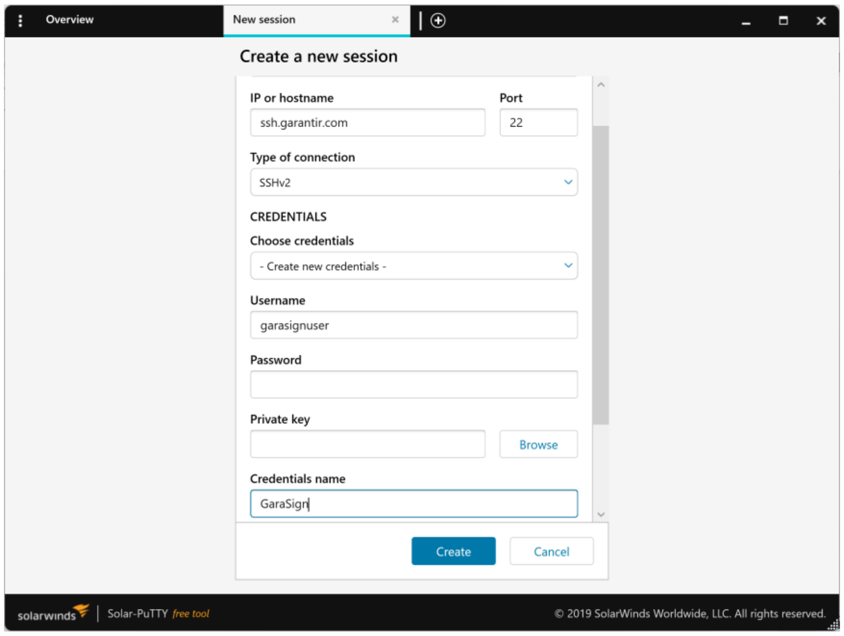
Stop the service
Restart the service

Description:
Agent to hold private keys used for public key authentication.

Name	Description	Status	Startup Type
Office 64 Source Engine	Saves install...		Manual
OpenSSH Authentication Agent	Agent to hol...	Running	Automatic
Optimize drives	Helps the co...		Manual
P9RdrService_d850d	Enables trig...		Manual (Trig
Parental Controls	Enforces par...		Manual
Payments and NFC/SE Manager	Manages pa...		Manual (Trig
Peer Name Resolution Protocol	Enables serv...		Manual
Peer Networking Grouping	Enables mult...		Manual
Peer Networking Identity Manager	Provides ide...		Manual
PenService_d850d	Pen Service		Manual (Trig
Performance Counter DLL Host	Enables rem...		Manual
Performance Logs & Alerts	Performance...		Manual
Phone Service	Manages the...	Running	Manual (Trig
Plug and Play	Enables a co...	Running	Manual
PNRP Machine Name Publication Service	This service ...		Manual
Portable Device Enumerator Service	Enforces gro...		Manual (Trig
Power	Manages po...	Running	Automatic
Print Spooler	This service s...	Running	Automatic
Printer Extensions and Notifications	This service ...		Manual
PrintWorkflow_d850d	Provides sup...	Running	Manual (Trig
Problem Reports Control Panel Support	This service ...		Manual

Solar-PuTTY

To use GaraSign with Solar-PuTTY, no additional configuration is required. Users can use Solar-PuTTY as they normally would, as shown below. If a popup dialog displays stating that Pageant is already running, this can be safely ignored.

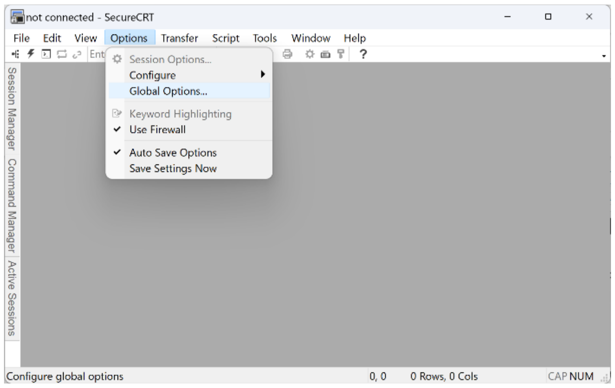


IP or hostname: ssh.garantir.com
Port: 22
Type of connection: SSHv2
Choose credentials: - Create new credentials -
Username: garasignuser
Password:
Private key:
Credentials name: GaraSign

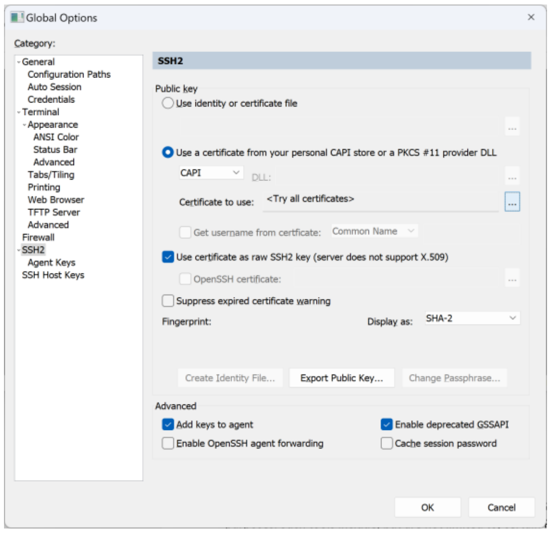
SecureCRT

To use GaraSign with SecureCRT, ensure that your desired key is configured in the Global Options. To do this, perform the following steps within SecureCRT:

1. Click on Global Options from the Options menu



2. Click the … button next to the *Certificate to use* label to select your desired certificate



SSH2

Public key

- ○ Use identity or certificate file
- ● Use a certificate from your personal CAPI store or a PKCS #11 provider DLL
- CAPI DLL:
- Certificate to use: <Try all certificates>
- ☐ Get username from certficate: Common Name
- ☑ Use certificate as raw SSH2 key (server does not support X.509)
- ☐ OpenSSH certificate:
- ☐ Suppress expired certificate warning
- Fingerprint: Display as: SHA-2

Advanced

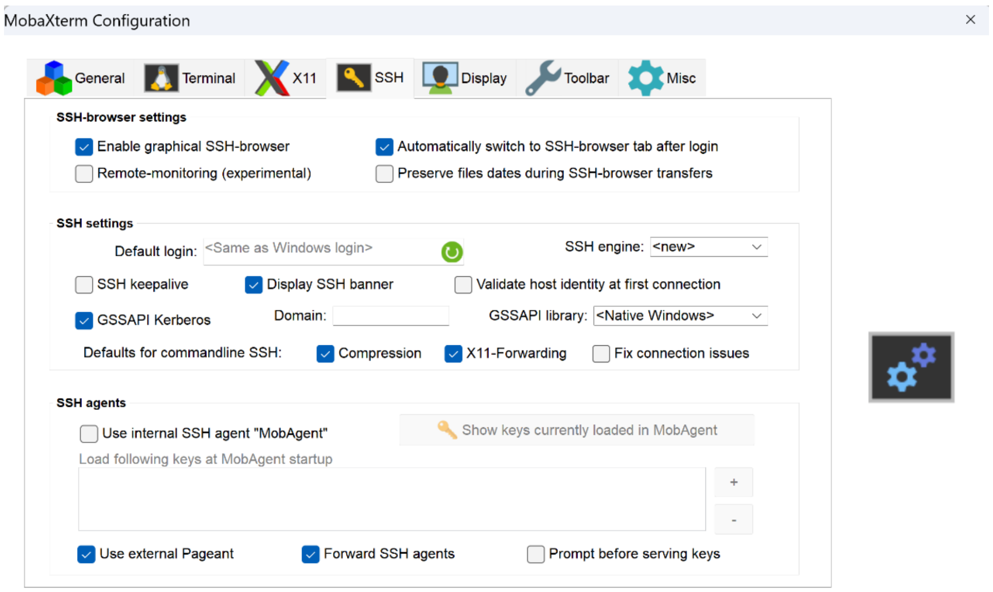
- ☑ Add keys to agent
- ☑ Enable deprecated GSSAPI
- ☐ Enable OpenSSH agent forwarding
- ☐ Cache session password

3. Select your desired certificate and click OK
4. Click OK

After the steps above have been performed, use SecureCRT as you normally would.

MobaXterm

To use GaraSign with MobaXterm, the *Use external Pageant* checkbox needs to be checked. Once this checkbox is checked, proceed to use MobaXterm as you normally would.



MobaXterm Configuration

SSH-browser settings

- [✓] Enable graphical SSH-browser
- [✓] Automatically switch to SSH-browser tab after login
- [ ] Remote-monitoring (experimental)
- [ ] Preserve files dates during SSH-browser transfers

SSH settings

Default login: <Same as Windows login>
SSH engine: <new>

- [ ] SSH keepalive
- [✓] Display SSH banner
- [ ] Validate host identity at first connection
- [✓] GSSAPI Kerberos

Domain:
GSSAPI library: <Native Windows>

Defaults for commandline SSH:

- [✓] Compression
- [✓] X11-Forwarding
- [ ] Fix connection issues

SSH agents

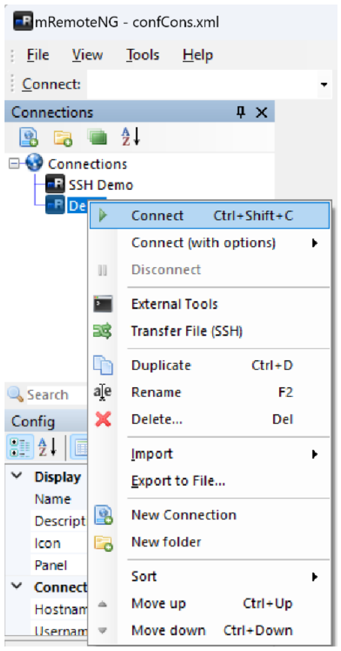
- [ ] Use internal SSH agent "MobAgent"

Load following keys at MobAgent startup

- [✓] Use external Pageant
- [✓] Forward SSH agents
- [ ] Prompt before serving keys

mRemoteNG

There is no additional configuration required to use GaraSign with mRemoteNG. Proceed to use mRemoteNG as you normally would.



FileZilla

There is no additional configuration required to use GaraSign with FileZilla. Proceed to use FileZilla as you normally would.

Snowflake

Applications accessing Snowflake via JDBC or ODBC may be able to use GaraSign keys for key-pair authentication, sometimes referred to as JWT authentication by some applications.

ODBC

The Snowflake ODBC driver v3.0.1+ links with OpenSSL 3.x and can therefore make use of the GaraSign OpenSSL 3.x Provider. If the calling application provides its own *openssl.cnf* file, then that file must be modified to add the GaraSign OpenSSL Provider at the top (be sure to backup the file before modifying it). If the solution does not provide its own *openssl.cnf* file, then one must be created with the GaraSign OpenSSL Provider configured and an environment variable named *OPENSSL_CONF* must be created that points to the newly created *openssl.cnf* file (be sure to create the environment variable such that the application can read it at runtime).

Once the OpenSSL configuration file and environment variable are both set, export an obfuscated key (this will not contain the real private key bytes) as described in the sections above and demonstrated in the example below:

```
export OPENSSL_CONF= /path/to/openssl.cnf
openssl pkey -in name:<insert key name> -out /path/to/output/file
```

The final step is to modify the ODBC connection string used by the application to point the *PRIV_KEY_FILE* attribute to the path of the obfuscated key file (i.e., the output of the *openssl pkey* command above).

Python

Python applications can leverage GaraSign's PKCS#11 library via the GaraSign-provided *PKCS11RSAPrivateKey* class defined in *pkcs11_rsa.py*. Once imported, the applications use this class as shown below:

```
import snowflake.connector as sc
from pkcs11_rsa import PKCS11RSAPrivateKey
 with PKCS11RSAPrivateKey('Garantir Token', '<insert GaraSign key name>') as private_key:
    conn_params = {
        'account': '<insert account number>',
        'user': '<insert username>',
        'private_key': private_key,
        'database': '<insert database>',
        'schema': '<insert schema>'
    }
     conn = sc.connect(**conn_params)
    cs = conn.cursor()
    cs.execute("SELECT CURRENT_VERSION()")
    one_row = cs.fetchone()
    print(one_row[0])
    cs.close()
    conn.close()
```

JDBC

Java applications that connect to Snowflake via JDBC can make use of either the PKCS#11 provider or pure Java JCE provider. This document describes the PKCS#11 provider approach but due to some limitations on the Snowflake driver end users may wish to use the pure Java JCE provider which is documented in its own user guide.

Perform the same steps as described in the Java section above, but additionally set three environment variables, some additional SunPKCS11 configuration, and ensure that the GaraSign PKCS#11 library is the first configured security provider.

Environment Variables:

GRS_PKCS11_FORCE_PRIVKEYS_NOT_SENSITIVE=1
GRS_PKCS11_FORCE_PRIVKEYS_EXTRACTABLE=1
GRS_PKCS11_FORCE_DUMMY_PRIVKEYS_MATERIAL=1
GRS_PKCS11_FORCE_CREATED_OBJECTS_ON_TOKEN=1

Additional SunPKCS11 Configuration:

Specify the following when configuring the SunPKCS11 provider:

```
ps.println("disabledMechanisms = {");
ps.println("    CKM_AES_CBC");
ps.println("    CKM_AES_CBC_PAD");
ps.println("    CKM_AES_KEY_GEN");
ps.println("    CKM_ECDH1_DERIVE");
ps.println("    CKM_MD5");
ps.println("    CKM_SHA_1");
ps.println("    CKM_SHA256");
ps.println("    CKM_SHA384");
ps.println("    CKM_SHA512");
ps.println("    CKM_RSA_PKCS_KEY_PAIR_GEN");
ps.println("    CKM_EC_KEY_PAIR_GEN");
ps.println("}");
```

Ensure GaraSign is the 1st Provider:

Security.insertProviderAt(pkcs11Provider, 1);

Use the GaraSign Key

Once instantiated, retrieve the desired key and pass it to the connection properties as “privateKey”, as shown below:

```
prop.put("privateKey", getPrivateKey());
Connection conn = DriverManager.getConnection(url, prop);
```

Other Integrations

There are other tools and applications that make use of CNG KSPs that *could* potentially use this KSP for signing purposes. Such tools include, but are not limited to, certain PDF creation software, Open Office tools on Windows, and Active Directory Certificate Services (ADCS). Please consult the documentation of those tools for further information.

Advanced Usage

This KSP supports the following advanced features: Approval Workflow, Hash Printing, and Authentication/Signature Tagging. Some of these features are triggered by environment variables. Since it may not always be desirable to use these features on every signature request, or at least not use the same data within these features on each signature request, it is recommended that these environment variables are set per process, instead of per user or globally on the computer.

Approval Workflow

Approval workflows are dictated by the signing key, not by configuration values. When a signing request is made with a key that requires approvals, this KSP will wait until the necessary approvals (or a rejection) are completed. In the background, the KSP periodically polls the server to check the status of the approvals. Once the approval process is complete, the KSP will automatically complete processing with either a valid signature or a failure.

Hash Printing

Most, if not all, signing applications hide the hash value to be signed from the end user. In some cases, it is desirable to know the hash value to be signed. For example, to verify a signing request during an approval workflow, an end user may want to check the hash to be signed against their own copy of the data to be signed. To facilitate this, this KSP supports Hash Printing which will print the hash to be signed to standard output (STDOUT) or to a specified file based on the values set in certain environment variables.

To print the hash to sign and continue signing data, create an environment variable named *garantir_ksp_print_hash_to_sign* with a value of either *STDOUT* or the full file path of the desired output file. If the environment variable is set to *STDOUT* the string "*Hash to sign:*" followed by the base 64 encoded hash to sign will be printed to standard output (with a line feed character prepended and appended to make it easy for the user to find the hash value). If the environment variable is set to a different value, the KSP will attempt to write the base 64 encoded hash value to that file (without the "*Hash to sign:*" prefix). After printing the hash to STDOUT or file, the signing process will continue.

To print the hash to sign and then terminate processing (i.e., do **not** sign the file), create an environment variable named *garantir_ksp_print_hash_to_sign_and_exit* with a value of either *STDOUT* or the full file path of the desired output file. If the environment variable is set to *STDOUT* the string "*Hash to sign:*" followed immediately by the base 64 encoded hash to sign will be printed to standard output. If the environment variable is set to a different value, the KSP will attempt to write the base 64 encoded hash value to that file (without the "*Hash to sign:*" prefix). After printing the hash to STDOUT or file, the process will be terminated, and no signature will be generated.

Note 1: When writing the hash to file, the original content of the file, if any, will be overwritten.

Note 2: When attempting to write the hash to file, if the file cannot be written to the KSP will terminate processing, even if the environment variable used is *garantir_ksp_print_hash_to_sign*.

Note 3: If both environment variables are specified, the *garantir_ksp_print_hash_to_sign_and_exit* environment variable value will be used and the other ignored.

Authentication/Signature Tagging

In certain situations, it may be desirable to send additional information to the GaraSign server beyond what the signing application may support. For example, it may be desirable to tag a signature request with the build number or audit hash of the file to be signed or send proof of an anti-virus scan during authentication. To facilitate this, this KSP supports sending arbitrary key-value pairs of information during Authentication and Signature requests.

To send key-value pairs during requests, create an INI file with at least one of the following sections: *auth* and/or *sign*. Place the key-value pairs needed for authentication in the *auth* section and place the key-value pairs needed for signing in the *sign* section. Then, create an environment variable named *garantir_cng_provider_params_file* and give it a value of the full file path to the INI file containing the key-value pairs. When a signing or authentication process is triggered, the appropriate key-value pairs will be sent to the GaraSign server.

Note 1: While the *auth* key-value pairs are sent during authentication, the default implementation of GaraSign does **not** do anything with them and does **not** store them. Organizations that implement custom authenticators server-side will be able to use these key-value pairs but in all other situations these values are ignored. This is **not** the case with signing key-value pairs. Signing key-value pairs are persisted with the signing request.

Note 2: If session token caching is enabled and the cached token has not expired, the *auth* key-value pairs will **not** be sent since authentication will be skipped.

Note 3: All key-value pairs in the INI file must exist before the KSP is called. This KSP will **not** automatically fill in any values.

Note 4: Organizations are free to come up with their own key-value pairs provided that the key name does **not** start with the word “garantir”. However, the following names are suggested for certain common scenarios:

Key	Value
pre_sign_audit_hash_<insert hash algorithm> **Examples:** **pre_sign_audit_hash_sha256** **pre_sign_audit_hash_sha384** **Note: The naming convention here is that sha256 is equivalent to sha-2-256. Organizations using sha3 shall use sha-3-X (e.g., sha-3-256).**	The base 64 encoded value of hashing the entire file with the specified hash algorithm before a signature is applied.
pre_sign_audit_file_size	The size (in bytes) of the entire file before a signature is applied.
build_id	The ID of the build that triggered this signing request.
build_system_id	For environments that have (or may have in the future) multiple build systems, this is the unique identifier of the build system triggering the signing request. This is **not** intended to be used to identify the host system that is triggering the job.

file_to_sign	The name of the file to sign, **not** including the folder path.
customer_signature_request_id	An identifier, intended to be globally unique, that the client can provide for the given signature request. While this is intended to be unique, this is **not** enforced.
release_identifier	Intended for signatures of code ready to be released. Value to be determined by each organization but a standard naming convention should be followed for consistency. For example, an organization may use something like *Q1ProductXYZ2018* to indicate that this is the release of Product XYZ in Q1 of 2018.
sign_command	The command used to call the signing application. This is intended to capture all the switches (i.e., arguments) given to the signing command. The calling application is automatically passed by the KSP, so it is not necessary to specify that (see note 5 below).
calling_application_version	The version number of the calling application. For example, this could be the version of signtool.exe.

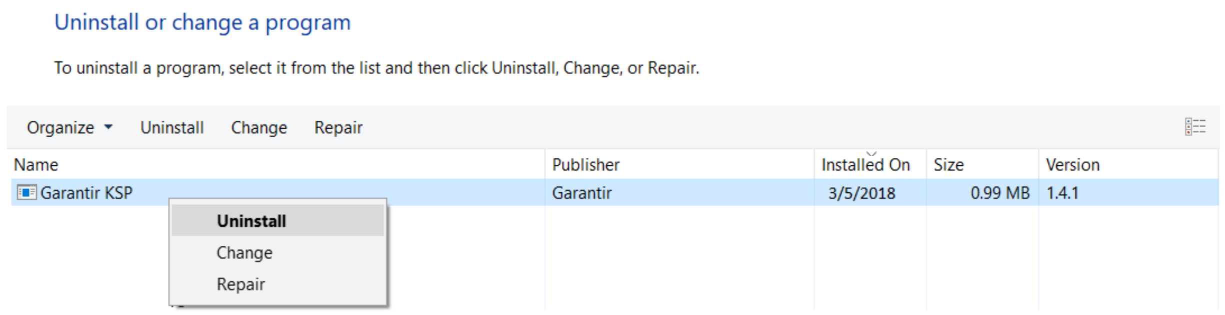
Note 5: It is **not** necessary to create a *sign* key-value pair that specifies the calling application (e.g., signtool.exe) as this is automatically handled by the KSP.

Uninstall

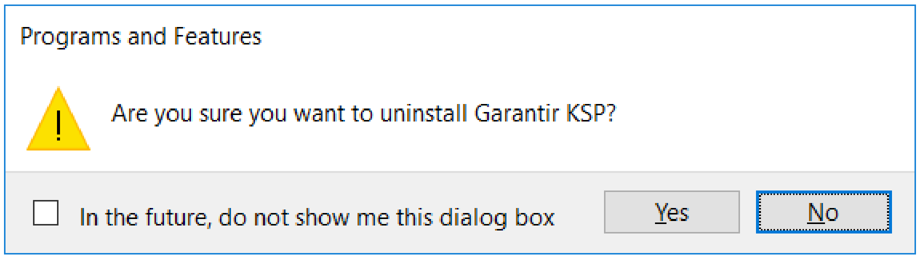
The uninstaller will remove the KSP's DLL and relevant installed artifacts, but it will **not** remove any certificates loaded to a user's Personal certificate store. Use *CertificateLoader* (with the */unload* switch) to remove these certificates. Since *CertificateLoader* is removed as part of the uninstallation process, this action must be done (for each applicable user account) prior to uninstalling the KSP, if unloading certificates is desired. See the Unloading Certificates section for more information.

To uninstall the KSP, perform the following steps:

1. Navigate to the *Add or remove programs* section of the Control Panel, right-click the *Garantir KSP* row, and click *Uninstall*.



2. Click Yes on the confirmation popup dialog.


Programs and Features
Are you sure you want to uninstall Garantir KSP?
In the future, do not show me this dialog box
Yes
No

3. Click Yes to allow the app to make changes to your device and wait for the processing to complete.

Frequently Asked Questions

How do I sign X?

If you are unsure of how to sign a particular artifact and cannot find it in this document, try the garasign wrapper tool. To use it, open a command prompt as your regular user and then execute *garasign help*. Follow the instructions from the output to learn how to use it for your use case.

Will this KSP allow me to timestamp data that I sign?

Yes, when using this KSP with tools like *signtool.exe*, you will still be able to timestamp data. This KSP plays no role in the timestamping process as that is all handled by the calling application (in this example, *signtool*) and your TSA provider. In fact, this KSP will not prohibit you from using any other features of *signtool* (or the other tools discussed in this document) as its functionality is only called when the actual data signing needs to occur.

Am I still satisfying my requirement to use an HSM?

The short answer is yes. The GaraSign server that this KSP communicates with sits in front of your company's cryptographic tokens. In most cases, these tokens are HSMs although GaraSign supports devices other than HSMs as well. Contact your GaraSign administrator to find out more about your organization's architecture and cryptographic tokens.

Will I still be able to use the Windows Certificate Store when using this KSP?

Yes. This KSP uses the Windows Certificate Store to store your certificates and maps them back to their unique identifier given by the GaraSign server (note: the private key material is **never** given to the client). When a signing request occurs against a given certificate from the Windows Certificate Store, this KSP translates that back to a request against the appropriate unique identifier to the GaraSign server.

How do I view the thumbprint of a certificate?

Open the certificate in the Windows Certificate dialog (from the Windows Certificate Store you can do this by double clicking on the entry). In the popup dialog, click on the Details tab and then scroll down until you see the Thumbprint value. A screenshot of this is provided in the Client Certificate section.

How can I tell if my data is signed?

This really depends on what you are signing. You can verify code signed by *signtool.exe*, *sn.exe*, & *mage.exe*/*mageui.exe* via the appropriate verification switches for the tool. Please consult the Microsoft documentation on these tools for more information. Additionally, code signed by *signtool.exe* will have a *Digital Signatures* tab present when you right-click the binaries and click on *Properties*. From there, you can view the details of the signature (e.g., signing certificate, digital timestamp, etc.). Emails protected by S/MIME will have the appropriate S/MIME ribbon placed on them in *Outlook*. Data signed within Microsoft Office applications will present a digital signature notification in the application.

How do I request an application be added to the list of supported integrations?

Please contact your Garantir representative.

Troubleshooting

In general, most problems can be diagnosed by enabling logging and viewing the log file. See the Logging section for instructions on how to configure logging. Provided below is a list of common error messages one might see in an application, command prompt, or log file and some suggested ways to resolve each issue.

Cannot Locate CertificateLoader

Description: Executing the command *where certificateloader* displays the message "INFO: Could not find files for the given pattern(s)" or executing the command *certificateloader* displays the message "'certificateloader' is not recognized as an internal or external command, operable program or batch file.'"

Cause: Installation did not complete successfully.

Resolution: When installing from the command line, make sure the command prompt is run as Administrator and make sure the installer is the correct one for the given platform (i.e., x86 for 32-bit or x64 for 64-bit). When installing from the GUI, make sure that the user has sufficient permissions on the host system to perform installation and that the Installation Complete dialog is displayed.

CertificateLoader Fails to Run

Description: Executing the command *certificateloader* displays the message "NCryptEnumKeys failed. Error code: 0x8009002D No certificate has been loaded to the certificate store."

Cause: Configuration is incorrect, authentication credentials are invalid, and/or network access to the GaraSign server is unavailable.

Resolution: Ensure that each of the following is true:

1. The command prompt is being run as the standard user account and **not** as Administrator.
2. The config.ini file for the user account exists in *%USERPROFILE%/AppData/Roaming/Garantir/GRS/.* This should automatically happen as part of *CertificateLoader* running.
3. The *Name*, *Port*, and *URL* values in the *Server* section of config.ini are all valid. Additionally, all other specified values are valid data types (see the Configuration Values section for more information).
4. The authentication credentials in the *user1* section are valid. For username & password authentication this means that the username and password values exactly match (case-sensitive) the values given by your system administrator. For client-certificate authentication this means that the hash value provided maps to the Thumbprint value of a non-expired certificate that chains up to an appropriate CA that is in the user's Personal Windows Certificate Store. For Kerberos (i.e., Windows authentication) this means that the user account is on the domain and mapped to a user in the GaraSign user database.
5. No network or operating system firewall is blocking access to the GaraSign server over the configured port. Use telnet or your web browser to verify this.

6. The SSL/TLS certificate of the GaraSign server (or load balancer, if applicable) is valid and trusted by the client computer. If the certificate is not trusted, you may need to import the CA certificate chain into the Intermediate Certification Authorities and/or Root Certification Authorities certificate store(s). Please consult your system administrator before modifying the trusted certificates on your computer.

CertificateLoader Does Not Load Any Certificates

Description: Executing the command *certificateloader* displays the message "No certificate has been loaded to the certificate store" but an error code is **not** displayed.

Cause: The user account is authenticating successfully but the account has **not** been assigned access to any keys.

Resolution: Contact your GaraSign system administrator to request access to the appropriate signing key(s).

Cannot see a Certificate I was Given Access to

Description: A GaraSign administrator has granted an account access to a signing key, but the corresponding certificate does **not** show up in that account's Windows Certificate Store.

Cause 1: *CertificateLoader* has not been run since permission to the key was granted.

Resolution 1: Manually run *CertificateLoader* to repopulate the Windows Certificate Store. See the Loading Certificates section for more information.

Cause 2: Caching is enabled and the current (non-expired) cache was created before access to the key was granted.

Resolution 2: Clear the cache and then rerun *CertificateLoader*. See the Clearing the Cache section for more information.

Mage.exe Fails with Selected Certificate

Description: Executing the command *mage.exe -sign <your manifest> -certhash "<your certificate>"* displays the message "This certificate cannot be used for signing … Internal error, please try again. Invalid provider type specified."

Cause: You are using a version of Mage.exe that is older than version 4.6.2. Mage versions prior to 4.6.2 do **not** support the CNG API.

Resolution: Use version 4.6.2 or later of Mage.exe.

Note: Even if you have a version of the .NET Framework or Runtime that is 4.6.2 or later, that does **not** mean that your .NET Tools are 4.6.2 or later. The .NET tools are distributed as part of the .NET Framework Developer Pack, **not** the .NET Framework itself.

MageUI.exe does not Display my Certificate

Description: My Certificate does **not** show up in the MageUI certificate selection box.

Cause: There are two possible causes for this:

1. You are using a version of MageUI.exe that is older than version 4.6.2. MageUI versions prior to 4.6.2 do **not** support the CNG API.

2. The attributes of your certificate (e.g., Key Usage, Extended/Enhanced Key Usage, etc.) do not qualify it for MageUI.

Resolution: Use a version of MageUI.exe that is 4.6.2 or later. If that does not resolve your issue, ensure your certificate has the correct attributes set for use with MageUI. See the Cannot Select my Certificate section for more information on certificate attributes.

Note: Even if you have a version of the .NET Framework or Runtime that is 4.6.2 or later, that does **not** mean that your .NET Tools are 4.6.2 or later. The .NET tools are distributed as part of the .NET Framework Developer Pack, not the .NET Framework itself.

VBA Macro is Signed with MD5

Description: The signature generated on the VBA Macro uses MD5 as its underlying hash algorithm.

Cause: As of the time of writing this document, VBA Macros only support digital signatures generated using MD5. This is **not** a limitation or bug of the KSP but rather a limitation forced by VBA.

Resolution: Continue to sign using the existing MD5 scheme. Where applicable, sign the entire Microsoft Office document in addition to the Macro. When sending the document to someone, sign the email with S/MIME. In advanced environments, consider utilizing a detached PKCS#7 signature as well. Use a stronger hashing algorithm for each of these other signatures.

Cannot Select my Certificate

Description: An application that makes use of CNG APIs does not display one or more certificates from your Personal Windows Certificate Store.

Cause 1: Some applications filter certificates by their purposes. In this case, the desired certificate does not have the appropriate purpose for the application.

On Windows, certificate purposes are dictated by the key usage, extended key usage (referred to as enhanced key usage on Windows), and configured purposes for the given certificate. The first two, key usage and extended key usage, are dictated by the certificate itself. These attributes are sealed in the certificate when the CA creates the certificate. The third item, the configured certificate purpose(s), is manually set for each certificate in the Certificate Store. To see the configured purposes for a certificate, open *certmgr.msc,* right-click the certificate and click Properties, then view the configured purposes under the General tab.

Resolution 1: If the certificate's purpose is restricted due to its configured certificate purpose, change the certificate purpose to the appropriate one via *certmgr.msc* and then refresh the certificate list in your application (may require the application to be restarted). If the certificate purpose is restricted due to the certificate's key usage or extended key usage attributes, a new certificate will need to be issued with the correct attributes.

Cause 2: In some cases, if the certificate does **not** chain up to a trusted root, the certificate is not displayed.

Resolution 2: Ensure that the certificate in question is trusted on your system by ensuring that the intermediary and root CA certificates in its chain are in the appropriate certificate stores on the target computer. **Always exercise caution when manipulating the trust stores on a computer**.

Signing Takes Longer Than Expected

Description: Signing performance is not as good as witnessed when demoed by Garantir.

Cause 1: Caching is **not** enabled which results in a new authentication request on each signing request.

Resolution 1: Turn caching on and set the TTL to a high value to reduce the number of authentication requests made by the KSP. See the Cache section for more information.

Cause 2: Logging is enabled which results in more file I/O on each signing request.

Resolution 2: Turn logging off and only enable it when you need to troubleshoot issues. See the Logging section for more information.

Cause 3: Not enough available resources. A non-trivial amount of the time spent processing is in calculating the hash locally and sending data over the network. These operations require system resources which may be currently allocated to other applications or processes. We have observed the mere presence of having certain browsers open or other applications running causes the signing request to take over three times as long as when those applications are closed.

Resolution 3: Close as many other applications and processes as possible before running your signing application. Also, use a computer with a fast CPU and lots of RAM, whenever possible.

Cause 4: There is a large distance between the GaraSign server and the client.

Resolution 4: Ask your GaraSign administrator about standing up a GaraSign server closer to your client to reduce the distance data must travel over the network.

Cause 5: The GaraSign server that you are connecting to has been put into debug mode (most likely for troubleshooting). In debug mode, there is more file I/O per request which can degrade performance.

Resolution 5: Ask your GaraSign administrator if the GaraSign server is in debug mode. If it is, run another test when debug mode has been turned off.

Appendix A

Glossary of Terms, Abbreviations, and Acronyms

Certificate Authority (CA)	An entity that issues digital certificates.
Cryptographic Application Programming Interface (CAPI)	The legacy cryptographic API included within Microsoft Windows operating systems.
Cryptography API: Next Generation (CNG)	The latest cryptographic API included within Microsoft Windows operating systems.
Cryptographic Service Provider (CSP)	A software library that implements CAPI.
Dynamic-Link Library (DLL)	A shared library for Microsoft Windows operating systems.
Elliptic-Curve Diffie-Hellman (ECDH)	A variant of the Diffie-Hellman protocol that uses elliptic-curve cryptography.
Elliptic-Curve Digital Signature Algorithm (ECDSA)	A variant of the Digital Signature Algorithm which uses elliptic-curve cryptography.
Graphical User Interface (GUI)	A user interface that allows users to interact with it via visual indicators and graphical icons.
Hardware Security Module (HSM)	A physical computing device that safeguards, manages, and makes use of cryptographic keys.
INI	An informal configuration file format.
Key Storage Provider (KSP)	A provider that allows an application to interact with asymmetric encryption keys.
Message Digest 5 (MD5)	A legacy cryptographic hashing algorithm.
Optimal Asymmetric Encryption Padding (OAEP)	A padding scheme standardized in PKCS#1 v2.
Public Key Cryptography Standards (PKCS)	A group of public-key cryptography standards published by RSA Security LLC.
Public Key Infrastructure (PKI)	A set of roles, policies, and procedures needed to create, manage, distribute, use, store, and revoke digital certificates and manage public-key encryption.
Registration Authority (RA)	The authority in a PKI that verifies requests for a digital certificate.
Rivest-Shamir-Adleman (RSA)	One of the first public-key cryptosystems.
Secure Hash Algorithm (SHA)	A set of cryptographic hashing with various output lengths.
Secure/Multipurpose Internet Mail Extensions (S/MIME)	A standard for public key encryption and signing of email data.
Transport Layer Security (TLS)	A cryptographic protocol designed to provide communications security over a computer network.
Time to Live (TTL)	A mechanism that limits the lifetime of data in a computer or network.