Managing certificates with IBM GSKit

An easy guide to creating, signing, installing, and using certificates with IBM Global Security Kit

Alexei Kojenov

November 06, 2012

This tutorial explains how to set up and use IBM Global Security Kit (GSKit) for typical certificate management tasks such as self-signed certificate generation, creation of a Certificate Authority (CA), requesting a certificate from a third-party CA, and installing certificates for use in SSL protocols.

Before you start

This tutorial describes how to use IBM GSKit and OpenSSL tools for common certificate management tasks. It is not a general tutorial on public key cryptography, X.509 certificates, or SSL/TLS.

About this tutorial

IBM Global Security Kit (GSKit) is a common component that is used by a number of IBM products for its cryptographic and SSL/TLS capabilities. While each product provides some minimal documentation on how to use GSKit, this tutorial provides a comprehensive, product neutral tutorial on how to perform common certificate management tasks.

The tasks in this tutorial are described with a command-line approach to ensure they can be incorporated into automation scripts.

Objectives

In this tutorial, you learn how to locate and set up the GSKit command-line utility, how to create different kinds of digital certificates, how to set up your own Certificate Authority and sign certificates, as well as how to install, use, and switch between certificates.

Prerequisites

This tutorial is written for system administrators, security specialists, and developers who use IBM products containing GSKit.

System requirements

You need an IBM product that includes GSKit version 7 or 8. You generally do not need administrative or root access to the system unless you need to install optional OpenSSL software. However, you need read and write access to the certificate key database of your product, which can require administrative or root privileges.

Overview

IBM Global Security Kit

IBM Global Security Kit (GSKit) is a library and set of command-line tools that provides SSL implementation along with base cryptographic functions (symmetric and asymmetric ciphers, random number generation, hashing, and so on) and key management.

The underlying cryptographic library, *IBM Crypto for C (ICC)*, is FIPS certified.

GSKit is used by many IBM software products for its security, usability, and FIPS certification. Some products expose and even require a user to use GSKit utilities for certain tasks, while others wrap GSKit's capabilities in their own interfaces.

GSKit availability

GSKit is a component and not a stand-alone product. It is **not** obtainable independent of the products that ship it. GSKit support and updates are provided as part of other products' support and updates.

Setup

Understanding GSKit installation methods and versions

GSKit supports two installation methods: global and local. Both types of installations may be present on a system at the same time.

- On a global installation, a single GSKit instance is shared by multiple products. In this configuration, GSKit libraries and executable files are placed in a common location on the system outside of the product's installation directory. If more than one product uses the same GSKit version, these products will not create multiple copies of GSKit, but instead share the single global copy.
- On a local installation, each product has its own, private version of GSKit. In this configuration, GSKit files are placed somewhere within the product's directory structure and their location may or may not be documented. If a global installation exists on the system, it is ignored by the product, which uses only its local installation of GSKit.

Different major versions of GSKit (for example, version 7 and version 8) are separate and can coexist as global installations.

This tutorial discusses GSKit versions 7 and 8 only, not any prior versions. All examples are given for GSKit 8. Unless noted otherwise, the same commands and options that are provided in the examples also work in version 7.

Finding GSKit on your system

The GSKit command-line tool is named as follows:

```
gsk<version>capicmd[_64]
```

where `<version>` is the GSKit major version (either 7 or 8). The `_64` suffix is added on 64-bit platforms.

To locate the GSKit installation on your system:

1. Read the product documentation for any guidance on locating and running GSKit. Usually, if a product requires running the GSKit command-line tool, it also documents its location. Some products may provide their own wrapper scripts that set the correct GSKit environment and pass the arguments to the correct instance of GSKit command-line tool. If this is the case, skip the rest of this section and the next section, "Configuring the environment to run GSKit", and use the script name instead of `gsk8capicmd_64`.
2. Determine if there is a global installation of GSKit:
   - On UNIX or Linux®, enter one of the following commands, `gsk7capicmd_64` or `gsk8capicmd_64`, on the command line. If anything other than an error message is returned, GSKit is installed and ready to use.
   - On Windows®, open Registry Editor and look for one of the following keys:
     `HKEY_LOCAL_MACHINE\SOFTWARE\IBM\gsk8\CurrentVersion\InstallPath`
     or
     `HKEY_LOCAL_MACHINE\SOFTWARE\IBM\gsk7\CurrentVersion\InstallPath`
     These keys indicate where GSKit is installed.
3. You can search the product's installation directories or the entire file system/disk for files and directories containing "gsk." There are two subdirectories, `lib` and `bin`, containing GSKit shared libraries and binaries.

Configuring the environment to run GSKit

The process to configure your environment to run GSKit varies depending on the type of platform you are using.

UNIX and Linux

For global installations of GSKit, no configuration is needed. The command-line tool is already on the executable path, and the libraries are in their standard system location. The GSKit commands can be run from any terminal window.

For local installations of GSKit, add its shared libraries directory to your environment:

```
export <Shared library path environment variable>=<GSKit library path>
export PATH=$PATH:<GSKit binary path>
```

The shared library path variable name depends on your platform:

Table 1. Shared library path environment variable name

Platform	Variable name
AIX	`LIBPATH`
HP-UX	`SHLIB_PATH`
Linux, Solaris	`LD_LIBRARY_PATH`

For example, to set the environment on Linux, use:

```
export LD_LIBRARY_PATH=/path/to/gskit/lib
export PATH=$PATH:/path/to/gskit/bin
```

Windows

Add both library and binary paths to the PATH environment variable. You can do this either in a command-line window for a single session, or change the global settings. To add the paths using a command line, type:

```
set PATH=C:\path\to\IBM\gsk7\bin;C:\path\to\IBM\gsk7\lib;%PATH%
```

Installing OpenSSL

Some tasks in this tutorial use OpenSSL. See Related topics for instructions on obtaining OpenSSL, and follow the OpenSSL instructions for installing it on your system.

Preparing a key database

GSKit stores public and private keys and certificates in a key database. A key database consists of a file with a `.kdb` extension and up to three other files with `.sth`, `.rdb`, and `.crl` extensions.

Your product may have already created a key database. If so, look at the product documentation to find its location. If you don't already have a key database, you need to create and initialize a new one.

To create and initialize a new key database, run the following command (depending on your version):

- Version 7:
  `gsk7capicmd_64 -keydb -create -db <filename>.kdb -pw <password> -stash`
- Version 8:
  `gsk8capicmd_64 -keydb -create -populate -db <filename>.kdb -pw <password> -stash`

The `-db` parameter indicates the file name for the new key database. The `-pw` parameter indicates the password to use to protect the key database file. The `-populate` parameter in version 8 is optional and tells GSKit to populate the key database with a number of predefined trusted CA

certificates. Version 7 always populates the new key database with the predefined trusted CA certificates. The `-stash` parameter tells GSKit to save the specified key database password locally in the .sth file so that it doesn't have to be entered on the command line in the future.

In the example scenarios in this tutorial, the following key database names are used:

- `server.kdb`: Server key database
- `client.kdb`: Client key database
- `ca.kdb`: Certificate Authority key database

Managing self-signed certificates

Creating a self-signed certificate

A self-signed certificate consists of a public/private key pair and a certificate for the public key that is signed by the private key. It is also known as a "root" certificate because it can be used to create a Certificate Authority.

Self-signed certificates can also be used in simple scenarios when both the client and the server are known to each other and can exchange certificates securely out-of-band.

To generate a self-signed certificate and store it in the key database, use the following command:

```
gsk8capicmd_64 -cert -create -db server.kdb -stashed -dn
"CN=myserver,OU=mynetwork,O=mycompany,C=mycountry" -expire 7300 -label "My self-signed
certificate" -default_cert yes
```

The `-db` parameter specifies the key database where the self-signed certificate should be stored. The `-dn` parameter specifies the distinguished name to use on the public key certificate. The `-expire` parameter indicates the number of days the certificate is valid. The `-label` parameter is a name to use for the self-signed certificate within the key database. The `-default_cert` parameter makes the newly created certificate the default and is an optional parameter.

Installing the certificate on client systems

For the clients to trust a certificate, its public part needs to be distributed to the clients and stored in their key databases. The process for doing this is:

1. Extract the public part to a file using the following command:
   ```
   gsk8capicmd_64 -cert -extract -db server.kdb -stashed -label "My self-signed
   certificate" -format ascii -target mycert.arm
   ```
   The `-db` parameter specifies the server key database that contains the certificate to be shared with clients. The `-label` parameter specifies the certificate's label within the key database. The `-target` parameter specifies the file name where the exported certificate should be stored.
2. Distribute `mycert.arm` to the clients.
3. Add the new certificate to the clients' key database as follows:

```
gsk8capicmd_64 -cert -add -db client.kdb -stashed -label "Server self-signed
certificate" -file mycert.arm -format ascii -trust enable
```

The `-db` parameter specifies the name of the client's key database file. The `-label` parameter specifies the label to be used for the certificate inside the key database file. The `-file` parameter specifies the file containing the certificate to be imported.

Creating a Certificate Authority (CA)

Creating a CA using GSKit

1. Initialize the CA key database and create the CA certificate. For example:
   ```
   gsk8capicmd_64 -keydb -create -db ca.kdb -pw mypass -stash
   gsk8capicmd_64 -cert -create -db ca.kdb -stashed -dn CN=CA,O=CA,C=US -expire 7300 -
   label "CA cert" -default_cert yes -ca true
   ```
   The `-db` parameter specifies the file name to be used for the CA's key database file. The `-pw` parameter specifies the password to use to protect the key database file. The `-expire` parameter specifies the number of days before the certificate expires. The `-dn` parameter specifies the distinguished name to use on the CA certificate. The `-label` parameter specifies the name to be used for the CA certificate in the key database file.
2. Extract the CA's root certificate. This certificate must be installed at both the clients and servers:
   ```
   gsk8capicmd_64 -cert -extract -db ca.kdb -stashed -label "CA cert" -format ascii -
   target ca.arm
   ```
   The `-db` parameter specifies the file name of the CA's key database file. The `-label` parameter specifies the CA's certificate label in the key database file. The `-target` parameter specifies the file that is stored in the exported CA certificate.

Issuing a server certificate with a CA

For clients to verify a server's identity, the CA must issue a signed server certificate to the server.

1. The CA's root certificate must be added to the server's key database and marked as trusted, as follows:
   ```
   gsk8capicmd_64 -cert -add -db server.kdb -stashed -label "My CA root" -file ca.arm
   -format ascii -trust enable
   ```
   The `-db` parameter specifies the name of the server's key database file. The `-label` parameter specifies the label to use for the CA's root certificate in the database file. The `-file` parameter specifies the file that contains the CA's root certificate.
2. At the server, create a server certificate request as follows:
   ```
   gsk8capicmd_64 -certreq -create -db server.kdb -stashed -label "My CA signed
   certificate" -dn "CN=host.mycompany.com,OU=unit,O=company" -file cert_request.arm
   ```
   The `-db` parameter specifies the name of the server's key database file. The `-label` parameter specifies the label to use for the server certificate in the key database file. The `-dn` parameter specifies the distinguished name to use on the certificate. The `CN` parameter specifies the DNS name of your server, which is necessary for an SSL client to validate the certificate.

You can also request a subject alternative name (SAN) extension by using `-san_dnsname` or `-san_ipaddr` options (not supported in version 7). For example:

```
gsk8capicmd_64 -certreq -create -db server.kdb -stashed -label "My CA signed certificate" -dn "CN=host.mycompany.com,OU=unit,O=company" -san_dnsname "host1.mycompany.com,host2.mycompany.com" -san_ipaddr "10.10.10.1,10.10.10.2" -file cert_request.arm
```

3. The certificate request must be transported to the CA, and the CA must sign the certificate as follows:

   ```
   gsk8capicmd_64 -cert -sign -file cert_request.arm -db ca.kdb -stashed -label "CA cert" -target cert_signed.arm -expire 364
   ```

   The `-file` parameter specifies the file that contains the certificate request. The `-db` parameter specifies the name of the CA's key database file. The `-label` parameter specifies the label of the CA's root certificate that should be used to sign the certificate request. The `-target` parameter specifies the file to be used for the signed server certificate.

   If a SAN extension was requested in the server certificate request, you can either use the `-preserve` option to keep the requested values or override them by specifying your own `-san_dnsname` or `-san_ipaddr` options with the `-sign` command (not supported in version 7). If you use both `-preserve` with `-san_dnsname` or `-san_ipaddr`, the values are merged with the ones requested. For example:

   ```
   gsk8capicmd_64 -cert -sign -file cert_request.arm -db ca.kdb -stashed -label "CA cert" -target cert_signed.arm -expire 364 -preserve -san_dnsname "host3.mycompany.com" -san_ipaddr "10.10.10.3"
   ```

   Note: At the time of this writing (GSKit version 8.0.14.22), there is a bug that generates invalid extensions when both `-preserve` and `-san_dnsname` or `-san_ipaddr` options are used. This bug prevents servers from receiving certificates that are signed with this combination of options. Avoid using `-preserve` until this problem is fixed.
4. The server must receive the signed certificate from the CA and set it as the default for communicating with clients as follows:

   ```
   gsk8capicmd_64 -cert -receive -db server.kdb -stashed -file cert_signed.arm -default_cert yes
   ```

   The `-db` parameter specifies the name of the server's key database file. The `-file` parameter specifies the name of the file that contains the signed server certificate.

Distributing the CA root certificate to clients

For your clients to validate the signed certificate that they receive from the server during an SSL connection, they must trust your Certificate Authority. This is achieved by installing the CA root certificate on the clients.

1. Transfer the CA root certificate to clients. (See the ca.arm file created above.)
2. Add the CA root certificate to the client key database and enable trust as follows:

   ```
   gsk8capicmd_64 -cert -add -db client.kdb -stashed -label "My CA root" -file ca.arm -format ascii -trust enable
   ```

   The `-db` parameter specifies the client's key database file to store the CA's root certificate. The `-file` parameter specifies the file that contains the CA's root certificate.

Using a third-party Certificate Authority

Install the CA root certificate

Instead of setting up its own certificate authority, a company may use a third-party certificate authority to sign its server certificates. The client and server must have access to the third-party CA's root certificate to verify the server certificates that are signed by the third-party CA.

GSKit ships with a collection of third-party root certificates from well-known CA companies, such as Thawte, Verisign, and Entrust. If the server is going to use one of these well-known companies to sign its certificates, this step can be skipped. But if the server is going to use certificates from a third-party CA whose root certificate is not shipped with GSKit, the third-party CA's root certificate must be imported to both the server and the clients' key database files as follows:

1. Obtain the CA root certificate. The process for this varies depending on the third-party CA's procedures. Third-party CAs often make their root certificates available for download.
2. Add the third-party's root CA certificate to both server and client key databases and mark it as trusted as follows:
   ```
   gsk8capicmd_64 -cert -add -db server.kdb -stashed -label "Some CA root" -file ca.der -format binary -trust enable
   gsk8capicmd_64 -cert -add -db client.kdb -stashed -label "Some CA root" -file ca.der -format binary -trust enable
   ```
   This example uses a third-party CA root certificate that is in a binary format. If the certificate is in an ASCII format, use the `-format ascii` option. The `-db` parameter specifies the name of the key database to import the third-party CA root certificate into. The `-label` parameter specifies the label to use for the third-party CA root certificate inside the key database file. The `-file` parameter specifies the file that contains the third-party CA root certificate.

Requesting a certificate using a signing request

In this scenario, GSKit creates a certificate request, the third-party CA signs the certificate in the request, and GSKit imports the signed certificate into the server key database.

1. Generate a server certificate request using the server's key database file:
   ```
   gsk8capicmd_64 -certreq -create -db server.kdb -stashed -label "Some CA signed certificate" -dn "CN=host.mycompany.com,O=company,C=country" -file cert_request.arm
   ```
   The `-db` parameter specifies the name of the server's key database file. The `-label` parameter specifies a label to refer to the newly created certificate in the key database file. The `-dn` parameter specifies the distinguished name to be used on the server's certificate. The `-file` parameter specifies the file to contain the exported certificate signing request. The `CN` parameter specifies the DNS name of your server. This is necessary for an SSL client to validate the certificate.
   You can also request SAN extension by using `-san_dnsname` or `-san_ipaddr` options (not supported in version 7). For example:
   ```
   gsk8capicmd_64 -certreq -create -db server.kdb -stashed -label "Some CA signed certificate" -dn "CN=host.mycompany.com,OU=unit,O=company" -san_dnsname
   ```

`"host1.mycompany.com,host2.mycompany.com" -san_ipaddr "10.10.10.1,10.10.10.2" -file cert_request.arm`

2. Send the certificate request (that is, the cert_request.arm file) to the CA. The process for submitting a certificate signing request varies among CA companies. Often the signing request can be submitted using a web form.
3. The CA then returns the signed certificate. In this scenario, the assumption is that the signed certificate is in a file that is called cert_signed.arm and is in an ASCII format.
4. Receive the signed certificate into the server's key database file and set it as the default for communicating with clients:
   `gsk8capicmd_64 -cert -receive -db server.kdb -stashed -file cert_signed.arm -default_cert yes`
   The `-db` parameter specifies the name of the server's key database file. The `-file` parameter specifies the name of the file that contains the signed certificate.

Requesting a certificate without a signing request

Some Certificate Authorities do not accept signing request files. Instead, they generate the signing request internally on behalf of the requesting server and then sign it as one transaction. The CA then returns to the server two files, one containing the private key for the server to use and one containing the signed server certificate. In this example, the assumption of the two files is as follows:

- host.mycompany.com.crt: This is the file that contains the signed server certificate.
- host.mycompany.com.key: This is the file that contains the server's private key

To use these files, they must be converted to an industry standard format called PKCS12 before they can be imported into a key database.

1. Use OpenSSL to convert the two files into a PKCS12 file as follows:
   `openssl pkcs12 -export -in host.mycompany.com.crt -inkey host.mycompany.com.key -out host.mycompany.com.p12 -name "CA signed"`
   The OpenSSL command prompts you to enter a password. This password is only used temporarily so it can be any arbitrary password. In this example, the password is set to `abc`. The `-in` parameter specifies the file that contains the signed server certificate. The `-inkey` parameter specifies the file that contains the server's private key.
2. Import the certificate from the PKCS12 file to the server's key database file as follows:
   `gsk8capicmd_64 -cert -import -db host.mycompany.com.p12 -pw abc -target server.kdb`
   The `-db` parameter specifies the name of the PKCS12 file. The `-pw` parameter specifies the password that protects the PKCS12 file. The `-target` parameter specifies the name of the server's key database file. You are prompted for the password that protects the target database file.
3. Make the imported certificate the default certificate to use for communications as follows:
   `gsk8capicmd_64 -cert -setdefault -db server.kdb -stashed -label "CA signed"`
   The `-db` parameter specifies the name of the server's key database file. The `-label` parameter specifies a label of the imported certificate.

GSKit security considerations

Protecting private keys

If an attacker obtains access to the private keys the associated certificates can't be trusted, compromising the servers that depend on them. You can help protect the key database file by:

- Using a strong password for your key database file.
- Protecting the stored password file (the .sth file) using the file system's security mechanisms if you use the GSKit stashed password feature. For example, you can set the file permissions to restrict access to this file to certain users.
- Restricting file system access to the key database file (the .kdb file) so that it is only readable by the users that run an application that uses the key database.

Verifying the identity of certificate requesters

If you manage your own Certificate Authority, you must ensure that any certificate signing request comes from an identity that is authorized to access the resource the requested certificate is for. The trustworthiness of certificates issued by the Certificate Authority is only as good as the process used to verify the identity of the requester.

Tips and tricks

Listing key database contents

To get a short list (labels only) of all certificates in a key database, use the following command:

```
gsk8capicmd_64 -cert -list -db server.kdb -stashed
```

The `-db` parameter specifies the name of the key database file.

To get detailed information about a particular certificate, use the following command:

```
gsk8capicmd_64 -cert -details -db server.kdb -stashed -label "My certificate"
```

In this command, the `-db` parameter specifies the name of the key database file. The `-label` parameter specifies the label of the certificate in the database.

Switching between certificates

A server's key database file can have multiple server certificates in it. However, only one certificate, which is known as the default certificate, can be used by the server at a time. You can change which certificate is the default certificate using the following command:

```
gsk8capicmd_64 -cert -setdefault -db server.kdb -stashed -label <certificate's label>
```

In this command, the `-db` parameter specifies the name of the server's key database file. The `-label` parameter specifies the label of the certificate to make the default.

Because client key database files have only the server certificates in them and no private keys, none of the certificates in a client key database can be set as the default.

Verifying the server certificate

Verifying the server certificate using a browser

If your GSKit key database file is being used to implement SSL on a web server, connect to the server with a web browser using an `https://server:port` syntax. You may get a security warning if your browser doesn't have the signing CA certificate or the self-signed certificate that is used by the server. But most browsers let you display information about the certificate currently being used by the server.

Verifying the server certificate using OpenSSL

1. Connect to the server and display the certificates using the `-showcerts` parameter of the OpenSSL as follows:
   `openssl s_client -connect server:port`
   The `-connect` parameter specifies the server domain name and the port that the server is listening on.
2. From the command's output, copy everything from "BEGIN CERTIFICATE" to "END CERTIFICATE," including those two lines. In this example, assume the command output is copied to a file called server.cert.
3. Use OpenSSL to display the certificate as follows:
   `openssl x509 -in server.cert -noout -text`
   The `-in` parameter specifies the name of the file that contains the output from the `-showcerts` command.

If you add the `-showcerts` option in step 1, you get the full certificate chain. You can repeat steps 2 and 3 for each certificate in the chain to analyze them.

Related topics

- Public key certificate is a good starting point to learn about digital certificates.
- Certificate authority Find out more about Certificate Authority.
- Transport Layer Security is a detailed overview of SSL/TLS protocols.
- OpenSSL is an open source project for implementing the SSL/TLS protocols and has certificate management tools that are referenced in this tutorial.