Tivoli software

IBM

IBM Global Security Kit 7.0: Managing certificates

White Paper

Ori Pomerantz (orip@us.ibm.com)

April 2010

Table of Contents

Introduction

About this paper

IBM® Global Security Kit (GSKit) secures communications automatically. However, the initial establishment of trust between two components cannot be automated and must be done manually.

This white paper is about the purpose of certificates and the steps that are required to manage them certificates correctly.

Audience

This paper is for implementors and system administrators who need to configure IBM software to communicate securely using encryption and certificates.

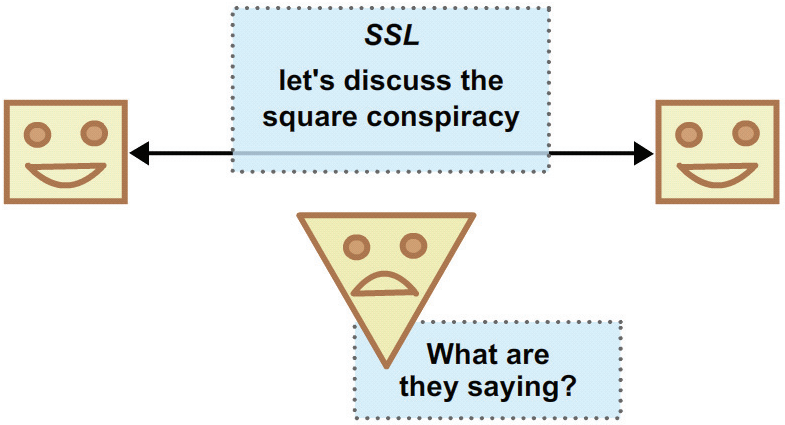
White paper

1 Why manage certificates

1.1 Protecting information in transit

GSKit uses secure sockets layer (SSL) to protect information in transit between two computers. It uses known, trusted algorithms such as AES and RSA to ensure confidentiality.

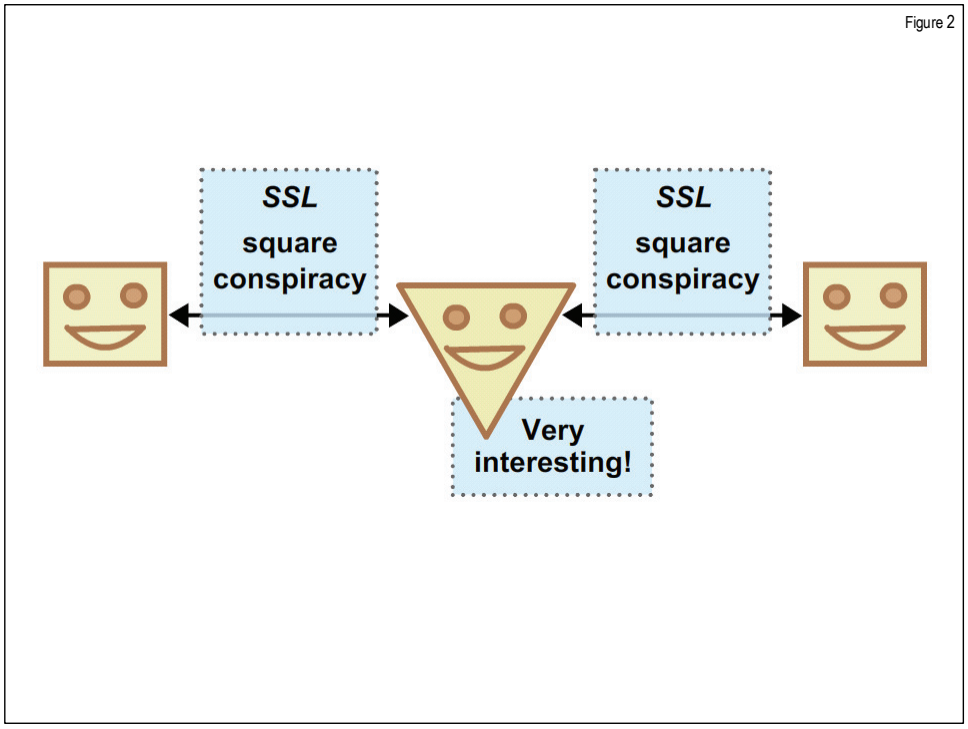
Figure 1



For example, in this illustration there is an SSL connection (called ***SSL tunnel***) between the two squares. Given current cryptographic knowledge, there is no way for the triangle in the middle to break the encryption and read the messages.

1.2 Man in the middle attacks

Unfortunately, there is one technique that might break SSL tunnels. Suppose the attacker participates at the start of the SSL tunnel and pretends to be the other participant. The attacker can eavesdrop on the connection.



For example, in this illustration the squares both have SSL tunnels and think they are communicating securely. However, they are both communicating securely **with the triangle in the middle**. The triangle is maintaining the charade by relaying the messages from one tunnel to another. This attack type is called a ***man in the middle***.

To prevent man in the middle attacks, SSL requires servers to present a cryptographic certificate, as explained in the next section. You can configure the server so that clients must present certificates.

2 Certificates

Participants use ***certificates*** to ensure the identity of the other participant and prevent man in the middle attacks. The certificates are integrated into the SSL tunnel so that an attacker cannot possibly relay the certificate from one side to another.

Each certificate contains two identities: the ***subject***, the entity the certificate identifies, and the ***issuer***, the entity that signs off on it.

Figure 3

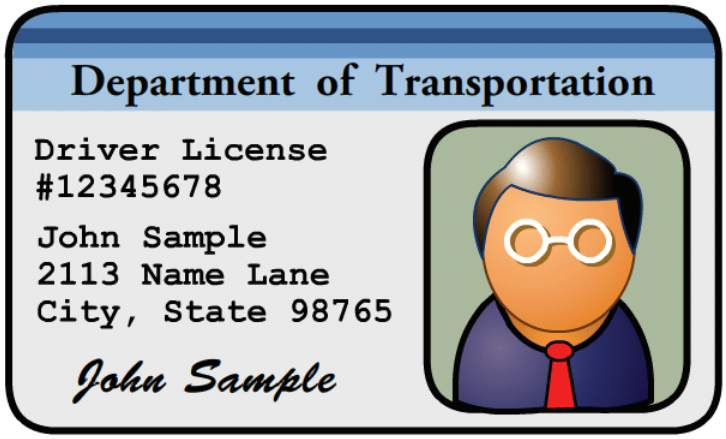
A certificate is like a driver's license. Just as a driver's license identifies a person, certificates identify an entity (typically a person or a computer). Just as a driver's licenses is issued by an authority, a certificate is issued by an authority. In the following table, you can see a comparison of a certificate and a driver's license for the same person.

Certificate		Driver's license	
Field	**Sample value**	**Field**	**Sample value**
Subject (LDAP distinguished name)	cn=Janice Sample,o=example	Driver	Janice Sample
Issuer (LDAP distinguished name)	cn=CA,o=example	Issuing Agency	Texas Department of Public Safety

2.1 Trusting certificates

Anyone can print a fake driver's license, like the one in this illustration, and anyone can issue a certificate.

Figure 4



To ensure that only legitimate certificates are accepted, GSKit requires that the certificate for the **issuer** be stored in the key file, in a section called, *signer certificates*.

2.2 Certificate types

There are two types of certificates: Certificate authority (CA) and self-signed certificates.

2.2.1 Certificate authority certificates

Certificate authority (CA) certificates are certificates that come from a certificate authority, which functions as the issuer. The issuer is a known, trusted entity.

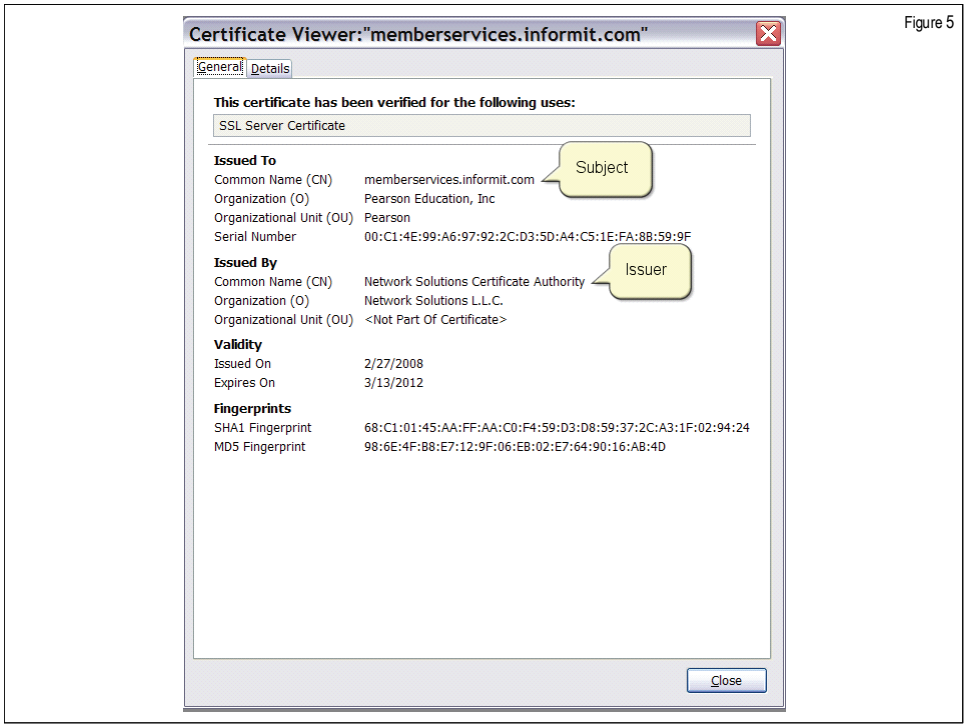


Figure 5

For example, in this certificate the issuer is Network Solutions. Network Solutions makes money by creating certificates. They have strong motivation to ensure that they only sign certificates that are valid. The subject of the certificate is **memberservices.informit.com**, a server that belongs to Pearson Education. You can use that server to buy IBM Press books.

A pre-existing relationship with Pearson Education is not needed to trust the certificate. If your browser is aware of the Network Solutions certificate authority, it accepts the certificate as valid.

2.2.2 Self-signed certificates

The issuer and subject of a ***self-signed certificate*** are the same. This certificate type is used for a certificate belonging to a certificate authority. Self-signed certificates are also used for internal servers, when the trouble and expense of getting a CA certificate is not necessary.

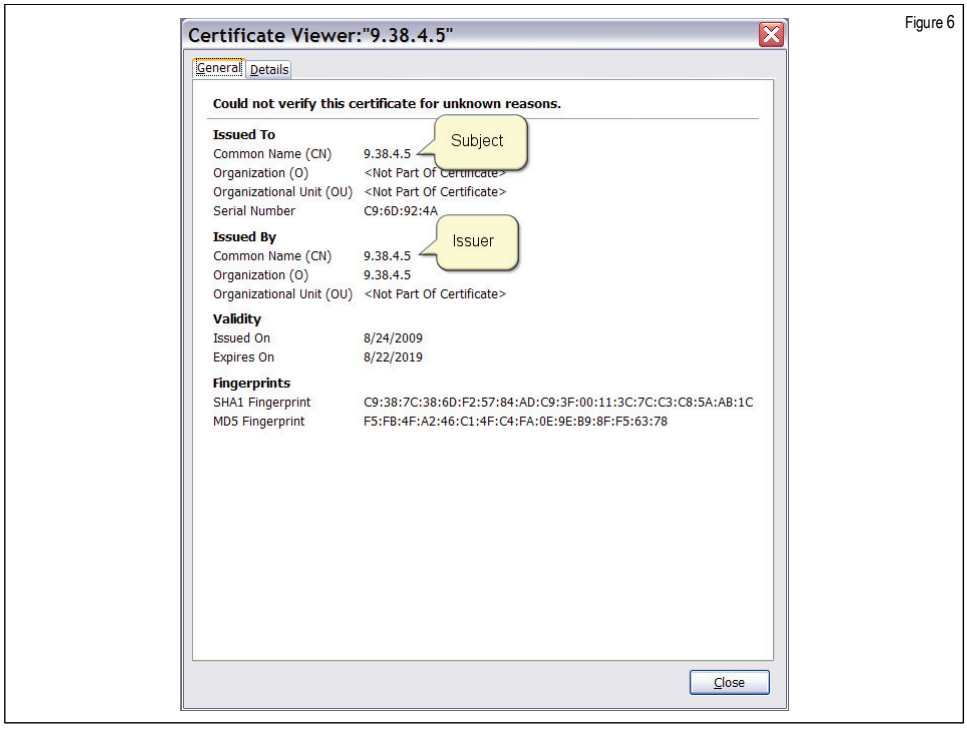


Figure 6

For example, this figure depicts a certificate for an internal server within IBM. All the people who use this server are IBM employees, and the information protected by the server is not sensitive. In this case, a CA certificate is not needed. The people who configured it generated their own self-signed certificate. As you can see, the subject and issuer are the same entity (IP address **9.38.4.5**).

2.3 Certificates and SSL tunnels

Note: This section gets a little deeper into how certificates work. This information is not required for day-to-day administration of software that uses GSKit.

Two pieces of information that are associated with a certificate are a private key and a public key.

The ***private key*** is stored by the subject of the certificate and must not be accessible to anybody else. This key is required to create a new SSL tunnel that is identified by the certificate. The signer's private key is also required to create new certificates.

The ***public key*** is available in the certificate. The public key is needed to verify the identity of the subject when creating a new SSL tunnel and to verify that the issuer of a certificate is indeed the one who issued it.

2.4 Certificate verification in tunnel creation

This process is used to verify a certificate when creating a new SSL tunnel.

1. The client and server create a TCP connection and attempt to initiate an SSL tunnel.
2. Side A (always the server, but possibly the client) sends the certificate to side B.
3. Side B attempts to validate side A's certificate. For the certificate to be verified and the tunnel created, several conditions must be fulfilled:
    a. The certificate of the issuer must be in side B's truststore. A ***truststore*** is a key file that contains the certificates of entities trusted by side B to identify other entities correctly.
    b. The certificate sent by side A must have a valid signature by the issuer.
    c. The certificate must not be on a ***certificate revocation list (CRL)***. A CRL is a list of certificates signed by an issuer but no longer valid. Typically, a certificate gets on a CRL when there is a risk that the private key for it was disclosed to unauthorized parties.
    d. During tunnel creation, side A used the private key that corresponds to the public key inside side A's certificate.

3 Key files

Secure communication requires entities to store certificates: their own certificates and the keys of trusted certificate authorities. Those certificates are typically stored in ***key files***, which can be manipulated using the **ikeyman** utility.

3.1 Key file types

3.1.1 Cryptographic Message Syntax

Cryptographic message syntax (CMS) is a file format used primarily by applications written in C and C++. CMS key files are typically split into several components, each with its own extension.

- **.kdb**: This file is the main file, which contains the actual certificates.
- **.rdb**: This file contains pointers and indexes to speed up data access.
- **.crl**: This file contains the certificate revocation list. Certificates are typically revoked because their private key was deleted or disclosed to unauthorized parties. Revoked certificates cannot be used by the key file.
- **.sth**: For security reasons, key files are encrypted with a password. However, key files that belong to servers must be opened without manual intervention. To do this, the password must be available to the server process.

  The password can be specified in cleartext in a configuration file, or in a ***stash file***. A stash file contains the password, protected by simple encryption. While a stash file does not protect against a determined attacker, it prevents casual snooping and inadvertent disclosure. To get the password to the key file, a person must purposely try to get it.

  The extension for stash files is **.sth**.

3.1.2 Java keystore

Java keystore (JKS) is a file format used in Java applications. From the perspective of GSKit, the two file types are similar. There are two differences:

- JKS files are stored in one file, which has a **.jks** extension.
- There is no stash file. The password is typically specified in a Java properties file.

3.2 Key file structure

Regardless of the type, key files have three sections: personal certificates, personal certificate requests, and signer certificates.

3.2.1 Personal certificates

Personal certificates are certificates for which the entity that owns the key file is the subject. These certificates contain a private key. They can be used to create SSL tunnels.

3.2.2 Personal certificate requests

For a certificate authority (CA) to sign a certificate, it must have the certificate's public key. However, the private key for the certificate must remain with the subject. The ***personal certificate request*** section contains the private keys for those requests.

3.2.3 Signer certificates

Signer certificates contain the public key of a certificate. An issuer's signer certificate is needed to verify that a certificate is valid.

4 Using GSKit

IBM education assistant (IEA) modules include explanations about using the graphical interface, **gsk7ikm** (also known as **ikeyman**). You can find these IEAs at the following URL:

http://publib.boulder.ibm.com/infocenter/ieduasst/tivv1r0/index.jsp?topic=/com.ibm.iea.gsk/gsk/7.0/using_ikeyman.html

The command-line interface, **gsk7cmd**, supports JKS by default. It can be configured to support CMS as well. This interface is documented in the following location:

http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.ihs.doc/info/ihs/ihs/cihs_ikeycmdline.html

There is a secondary interface, **gsk7capicm**, which is also available under Windows, and supports the CMS file type. It is documented in the following location:

ftp://ftp.software.ibm.com/software/webserver/appserv/library/v61/ihs/GSK7c_CapiCmd_UserGuide.pdf

Conclusion

Summary

You should now understand why certificate management is necessary, and understand enough of the theory behind it to manage certificates. To learn the practical steps to use certificates, see the IBM Education Assistant (IEA) modules about GSKit.

Resources

- To learn more about SSL, read the following Wikipedia article:

  http://en.wikipedia.org/wiki/Transport_Layer_Security

- The CMS key file standard is specified in RFC 3852, which is available at the following URL:

  http://www.ietf.org/rfc/rfc3852.txt

- To learn more about JKS, read the Java documentation:

  http://www.j2ee.me/j2se/1.4.2/docs/api/java/security/KeyStore.html