



HOST LINKS
G&R **SSL**

*Using SSL
for security
with G&R
products*

<http://www.gar.no/hostlinks/>

Microsoft, Windows, MS, MS-DOS are registered trademarks of Microsoft Corp.
IBM and PC are registered trademarks of IBM Corp.
UNIX is a registered trademark in the United States and other
countries, licensed exclusively through X/Open Company, Ltd.

Any other product names are trademarks of their respective owners.

Version 6.3
© Gallagher & Robertson as 1990-2005
All Rights Reserved

GALLAGHER & ROBERTSON AS, Kongens gate 23, N- 0153 Oslo, Norway
Tel: +47 23357800 • Fax: +47 23357801
www: <http://www.gar.no/>
e-mail: support@gar.no



Contents

- Secure communication 1**
- Secure Socket Layer..... 1
- Cryptographic algorithms..... 1
 - Symmetric ciphers..... 1
 - Asymmetric ciphers 2
- Hash algorithms..... 2
- Digital signatures..... 2
- Certificates 3
- Definitions..... 3
- Certificate Chains 4
- SSL Session Establishment 5
- Further documentation 5

- Glink for Windows 7**
- Windows SSL support..... 7
- Windows certificate store..... 7
- Glink and Windows certificates 8

- Glink for Java 9**
- SSL to the mainframe..... 9
- SSL to the Glink server 10

- Glink and SSL servers..... 11**
- Telnet 11
- G&R DSA gateway..... 11

- Ggate 13**

- Gweb 15**
- Functionality 15
- Browser to Web Server security..... 15
- The G&R Web Servers..... 16
 - GwebS..... 16
 - GwebSS..... 16



- G&R/SSL..... 17**
- Configuration 17
 - Basic terminology 17
 - Parameters for Windows servers..... 17
 - Parameters for UNIX/Linux servers 17
 - Keywords and command line formats..... 18

- Stunnel as an SSL server..... 21**
- Functionality 21
- Download and installation..... 21

- Certificates..... 23**
- X509 certificate content 23
 - Distinguished name fields 23

- Certificates supplied by G&R..... 25**
- Getting started with G&R certificates 25
 - Ggate 25
 - Gweb 25
 - Authentication of valid clients only 26
 - Glink 26

- Trusted third parties 27**
- Creating your own certificates 28
 - Using XCA to create a certificate request 28
 - Request a test certificate from Thawte 29

- OpenSSL license details 31**
- Original SSLeay license 32

Secure communication

This information is taken largely from the sources quoted in *Further Documentation* below. It is assembled here for your convenience.

Secure Socket Layer

SSL is a protocol layer that can be placed between a reliable connection oriented network protocol layer (e.g. TCP/IP) and the application protocol layer (e.g. HTTP, Telnet etc.). SSL provides for secure communication between client and server by allowing mutual authentication, the use of digital signatures for integrity and encryption for privacy.

Cryptographic algorithms

Symmetric ciphers

A symmetric cipher is an algorithm where the same key is used for both encryption and decryption. Typical key lengths span from 40 to 168 bits. Symmetric ciphers include: DES, RC2, RC4, RC5, Blowfish, IDEA and CAST. This unique key must be known to both sender and recipient, and be a secret between them for secure communication to take place.



Asymmetric ciphers

An asymmetric cipher is an algorithm where two separate keys are used for encryption and decryption. One key is defined as public and the other as private. These keys are generally much longer (256-4096 bits) than symmetric keys. Asymmetric ciphers include: RSA, DH and DSA. The public key of an individual or entity can be freely available to all senders, but only the owner knows the private key and can decode the message. Thus secure communication can take place without the sender and recipient sharing a secret key. In practice, because the asymmetric algorithms are so resource intensive, they are often used only to exchange a randomly generated one-time key (session key) used for symmetric encryption of the body of the message.

Hash algorithms

Hash algorithms are used to make message digests, which are short (typically 16-20 bytes) unique representations of arbitrary length messages. Hash algorithms include: MD2, MD5, SHA, SHA-1, RIPEMD, MDC2. Message digests are used to check the integrity of a message. If the recipient's own hash total of the message agrees with the digest, then the message content has not been changed.

Digital signatures

To prove that a message is from the designated sender it can be signed by the sender using the sender's private key. Anyone can decrypt the signature using the sender's public key, but only the designated sender could have encrypted it. The signature includes a hash digest of the message. This enables the recipient to check the message's integrity, and ensures that the same signature cannot be used for another message. The signature will normally also contain a unique sequence number that ensures that the sender cannot repudiate (deny having sent) the message.

Certificates

A certificate is used to identify a person or other entity, such as computer, and associate the entity with a public key. It contains the public key of the entity, and also information such as person/server name, validity period, algorithms used etc. To be of value, the certificate that identifies an entity and assigns it a public key must be issued by a trusted agency (*Certificate Authority*). The issuer of the certificate always signs it to prove that the trusted Certificate Authority really did issue it. If each party has a certificate that validates the other's identity, confirms the public key and is signed by a trusted agency, then they both are assured that they are communicating with whom they think they are.

Definitions

Public Keys

These are numbers associated with a particular entity, and are intended to be known to everyone who needs to have trusted interactions with that entity. Public keys are also used to verify signatures.

Digitally Signed

If some data is *digitally signed* it has been stored with the "identity" of an entity, and a signature that proves that entity knows about the data. The data is rendered unforgeable by signing with the entity's private key.

Identity

Identity means here a known way of addressing an entity. In some systems the identity is the public key, in others it can be anything from a Unix UID to an Email address to an X.509 Distinguished Name.

Signature

A signature is computed over some data using the private key of an entity (the signer).

Private Keys

These are numbers, each of which is supposed to be known only to the particular entity whose private key it is (i.e. it's supposed to be kept secret). Private and public keys exist in pairs in all public key cryptography systems. In a typical public key crypto system, such as DSA, a private key corresponds to exactly one public key. Private keys are used to compute signatures.

Entity

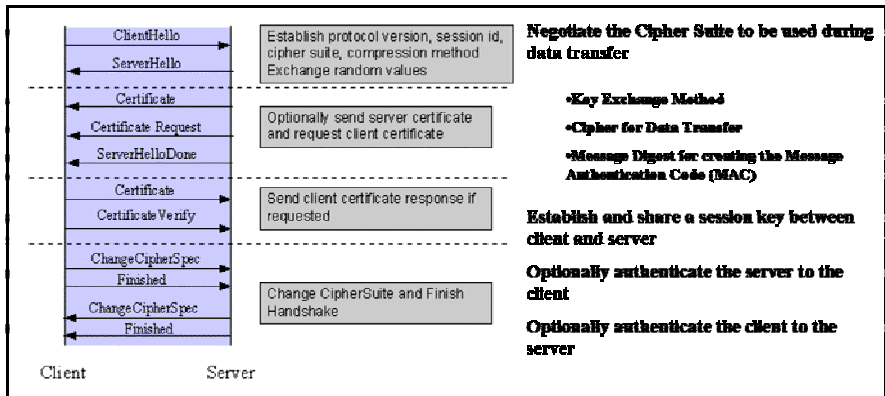
An entity is a person, organization, program, computer, business, bank, or something else you are trusting to some degree.

Basically, public key cryptography requires access to users' public keys. In a large-scale networked environment it is impossible to guarantee that prior relationships between communicating entities have been established or that a trusted repository exists with all used public keys. Certificates were invented as a solution to this public key distribution problem. Now a *Certification Authority* (CA) can act as a *Trusted Third Party*. CAs are entities (e.g., businesses) that are trusted to sign (issue) certificates for other entities. It is assumed that CAs will only create valid and reliable certificates as they are bound by legal agreements. There are many public Certification Authorities, such as VeriSign, Thawte, Entrust, and so on. You can also run your own Certification Authority using products such as the Netscape/Microsoft Certificate Servers or the Entrust CA product for your organization.

Certificate Chains

A Certificate Authority may issue a certificate for another Certificate Authority. When examining a certificate, you may need to examine the certificate of the issuer, for each parent Certificate Authority, until reaching one in which you have confidence. You may decide to trust only certificates with a limited chain of issuers, to reduce your risk of a "bad" certificate in the chain.

SSL Session Establishment



The session key referred to is the shared secret key for symmetric encryption between client and server. SSL 3.0 supports a choice of key exchange algorithms including the RSA key exchange when certificates are used, and Diffie-Hellman key exchange for exchanging keys without certificates and without prior communication between client and server.

Once an SSL session has been established it may be reused, thus avoiding the performance penalty of repeating the many steps needed to start a session. For this the server assigns each SSL session a unique session identifier that is cached in the server and which the client can use on forthcoming connections to reduce the handshake (until the session identifier expires in the cache of the server).

Further documentation

Some very good background information on SSL, encryptions, certificates etc. can be found in the documentations for ModSSL (an SSL implementation for Apache using OpenSSL). This information is located at:

- http://www.modssl.org/docs/2.8/ssl_intro.html

Sun provide an introduction to certificates at:

- <http://java.sun.com/products/jdk/1.2/docs/guide/security/cert3.html>



Glink for Windows

Windows SSL support

From Release 7.1 Glink supports SSL directly. You need no client software other than Glink itself. Glink uses the Secure Sockets Layer (SSL) included with Microsoft's Secure Channel (SChannel) security package to provide security-enabled communications including identity authentication and secure, private communication through encryption. All currently supported versions of Microsoft Windows include SSL and data encryption as a standard, and require no additional software.

Some older versions of Windows don't have the necessary security package versions installed for data encryption. On Windows 98 and Windows 95 clients, you may need to install the two Microsoft products below:

Internet Explorer 5.01 (or above), downloadable from MS Web site Directory Service Client (from a Win2000 server CD)

Windows certificate store

Because Glink bases itself on the Microsoft package, authentication of a server by Glink or authentication of Glink by a server is done using the Certificates stored in the PC's system certificate store. The certificate store can be viewed via the Internet properties dialog boxes either from the Control Panel or from Internet Explorer's tools menu.

Internet options/Content/Certificates...

The Microsoft view of server authentication tends to be focused on Web servers, and client authentication on browsers. Microsoft is also strongly focused on secure e-mail. Probably for these reasons, in addition to VeriSign, Thawte, Entrust and others that you might expect to find there, you will find hundreds of pre-installed Trusted Root Certificate Authorities on Windows platforms, and more arrive with each update of Internet Explorer. Whether or not you are aware of it, you trust all of these CAs to issue certificates that authenticate the identity of your correspondents.

For a detailed discussion of the Microsoft view as regards certificate management, try the general information at the Windows 2000 help site:

- <http://www.microsoft.com/windows2000/en/server/help/>
- Security/
 - Concepts/
 - Understanding security/
 - Public Key infrastructure/
 - Certificates overview

Glink and Windows certificates

Glink does not need certificates in order to use SSL for secure communication by encryption. The key exchange protocol ensures a negotiation of a secret key for symmetrical encryption for the duration of the session.

The server may optionally send its certificate during the handshake procedure, and optionally request a certificate from Glink. If Glink is configured to authenticate the server it uses the Windows interface, which checks the server certificate to see if it is valid and issued by a CA in the certificate store. If configured, Glink will verify that the server certificate's Common Name (CN=) corresponds to either the current server name or a specifically configured server name. The current server name is obtained by converting the host's IP address to a full domain path, and is displayed for your convenience.

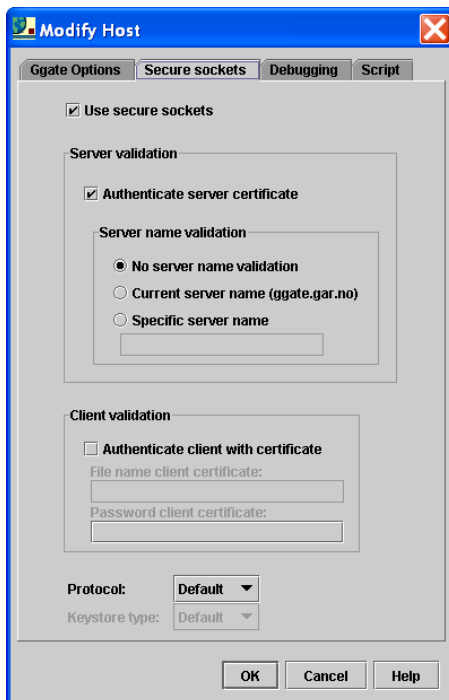
If the server has requested a client certificate Glink uses the configured client certificate, selected by name from a list of client certificates obtained from the Windows certificate store. Only installed certificates are displayed in the certificates list. If the host system is validating client certificates, you will need to specify the correct client certificate to use; otherwise the host will reject the certificate and disconnect the session. If no certificate is selected in the list box then the first one in the list will be used, otherwise if none are configured then a default certificate will be created.

Glink for Java

From release 6.5 Glink for Java supports SSL both on the connection to the mainframe to secure the dialog with the application, and on the connection to the Glink for Java server.

SSL to the mainframe

When configuring a Telnet or Ggate mainframe connection the configuration dialog box offers the choice of using SSL.



SSL to the Glink server

The server must be started using SSL parameters. The parameters are described in detail elsewhere in this document.

You configure SSL in the client using start-up parameters to Glink, running as an applet, as a Web Start application and as an application. You supply the parameters differently for the different modes, but the parameters are the same.

server.usessl	This parameter lets you configure Glink to use a secure connection to the Glink config server. Specify either 1 or 2. A value of 1 means use SSL without authenticating the server certificate. A value of 2 means use SSL and authenticate the server certificate. In most cases, you don't need additional SSL parameters. For example: <code>server.usessl=2</code>
server.valname	SSL optional parameter. If this option is used, Glink will in addition to authenticating the server certificate, verify that the server name (CN=) value contained in the received certificate corresponds to this value.
server.clientcert	SSL optional parameter. Specify the file path to the Key store that contains your client certificate. Necessary if the Glink server requests a client certificate.
server.clientcertpw	SSL optional parameter. Supply the password for the given Key store.
server.sslprotocol	SSL optional parameter. Sets the protocol to be used for the SSL connection, default value is TLS. Options: SSL Supports some version of SSL SSLv2 Supports SSL version 2 or higher SSLv3 Supports SSL version 3 TLS Supports some version of TLS TLSv1 Supports TLS version 1
server.keystoretype	SSL optional parameter. Specifies the Key store type. The default value is JKS. The other type supported is PKCS12.

Glink and SSL servers

Glink (Windows and Java) connects to hosts, gateways or front-ends that provide terminal sessions. SSL provides secure communication for any session protocol, running over any connection-oriented transport service. In the case of Glink these are terminal handling session protocols such as Telnet or *G&R/Ggate* running over TCP/IP networks.

The server side of the connection must offer SSL itself, or be reached via a server-side product that offers SSL-tunneling, i.e. the client connects to a port being serviced by the SSL-tunneling product, and it handles the SSL connection, but reroutes the session from itself to the server-side service that the client wishes to reach, after completing SSL security handshaking. The SSL-tunneling product remains as a conduit for the session, encrypting and decrypting the traffic.

Telnet

Because standard, insecure, Telnet is very widely spread, the most common solution to the Telnet security problem is to provide SSL on another port, and use SSL-tunneling to reach the standard Telnet server. A typical SSL-tunneling product for this usage is Stunnel, described later in this document. Telnet has been assigned a standard port (992) for secure Telnet connections (TelnetS). Unless Glink is configured otherwise it will connect to this port if you configure SSL for a Telnet session. The port can be overridden if the TelnetS service is on another port. SSL may well be offered on a non-standard port for Telnet variants such as TN3270, TN5250 and TNVIP.

G&R DSA gateway

This is the session protocol used to reach *Ggate*, the G&R gateway to Bull mainframes.

From Host Links Release 6.1 *Ggate* itself supports SSL; it requires no third party SSL-tunneling product to handle secure communications.



Ggate

Ggate is the G&R gateway to Bull mainframes used by G&R clients such as *Glink for Windows* and *Glink for Java*. Between *Ggate* and the mainframes the protocol is the native internal Bull mainframe protocol (DSA), running over either TCP/IP (using the RFC1006 interface to transport), or over OSI-transport. Between *Glink* and *Ggate* the protocol is G&R/Ggate over TCP/IP.

Ggate protocol is an extremely efficient protocol designed to map directly to the native internal protocol used by Bull hosts. All terminal handling is done in *Glink* itself, and the blocks transmitted to *Ggate* are already formatted as GCOS internal session protocol records. The mapping to native Bull internal protocol in *Ggate* is minimal, and from *Ggate* the connection to GCOS is direct, with no gateway or front-end required.

Ggate supports SSL directly, with no need for an SSL-tunneling product, and can act as an SSL server for secure *Glink* to GCOS communication.

You enable SSL support by adding the SSL parameters to the *Ggate* command line e.g.

```
gg_tcp -sslcf ggate.pem
```

The name of the file containing the server certificate to be used by *Ggate* is required. Other SSL parameters are optional. See elsewhere in this document for a detailed description of the available SSL parameters.



Gweb

Functionality

Gweb is the G&R Web gateway to mainframe applications running on Bull or IBM systems. The clients are standard browsers. The browsers see the mainframe applications as ordinary Web pages. The mainframe applications are completely unchanged, and see the browsers as ordinary terminals. The mapping from the terminal-specific presentation protocols used by the mainframes (Bull VIP7700, VIP7760, VIP7800, DKU7107 and DKU7211; IBM3270 and IBM5250), to the HTML presentation protocol needed by a browser, is done on the fly and automatically in Gweb. Gweb allows the administrator to facelift some or all of the mainframe terminal screens as required to make them more attractive to the clients.

Browser to Web Server security

When a browser is used to access any sensitive application running on a Web server, there is an obvious need for secure communications. SSL security is standard in browsers, and is standard in most Web servers. This need for secure communications is just as obvious when the Web server is being used as a gateway to reach sensitive mainframe applications via Gweb.

Gweb can be used together with any Web server that will run on a platform supported by the *G&R/Host Links* product range. If used with a standard Web server Gweb benefits from the standard SSL security between the browser and the Web server. Secure communication between Gweb clients and Gweb is available in the same way as for all other Web applications running on that server.

The G&R Web Servers

GwebS

The G&R Web server **GwebS** is a lightweight server, geared to efficiently meeting the specific needs of the G&R Web products. It is an alternative to a heavyweight, general purpose Web server that would require more disk space, time and expertise to install and configure.

GwebSS

From Host Links Release 6.1 a new version of the G&R Web server is included in the Gweb product. This new version of the server *GwebSS* supports SSL itself; it requires no third party SSL-tunneling product to handle secure communications.

This server is essentially the same product as **GwebS**, but adds HTTPS support. HTTPS is the secure version of the familiar HTTP (Hyper-Text Transport Protocol), and is used by browsers to connect to secure Web servers using SSL (Secure Socket Layer) for encryption and authentication. HTTPS has been assigned its own port for secure browser to Web server communication (443, rather than the standard HTTP port 80). Unless configured otherwise GwebSS will accept connects to this port. **GwebSS** requires at least a server certificate. e.g.

```
gwebss -sslcf gweb.pem
```

or the parameter:

```
SSLCertificateFile gweb.pem
```

can be added to the gwebs.XXX configuration file.

Other SSL parameters are optional. See elsewhere in this document for a detailed description of the available SSL parameters.

G&R/SSL

The SSL implementation used by G&R Server products is based on the OpenSSL toolkit. The details of the free license are included at the end of this document.

Whenever you install *G&R/Ggate*, *G&R/Gweb* or the *G&R/Glink for Java server* Release 6.1 or later, you also install *G&R/SSL*. There is no particular installation procedure.

Configuration

Basic terminology

- DER ASN.1 Distinguished Encoding Rules. DER is a binary format that is used to encode certificates and private keys.
- PEM Privacy Enhanced Mail. PEM is a text format that is the Base 64 encoding of the DER binary format. The PEM format also specifies the use of text BEGIN and END lines that indicate the type of content that is being encoded.

Parameters for Windows servers

The configuration dialog box allows entry of the various parameters. The names and functionality are consistent with the names used for the UNIX/Linux command line parameters described in detail below.

Parameters for UNIX/Linux servers

SSL parameters can be supplied in the command line e.g.

```
gg_tcp -sslcf ggate.pem           (start Ggate SSL)
gwebss -sslcf gweb.pem           (start GwebSS SSL)
```

In the case of GwebSS the parameters can also be placed in the gwebs configuration file in the format 'keyword' 'value':

```
UNIX      /usr/gar/config/default/
Windows  C:\gar\config\default
```

Command line filenames without a path will be looked for in <SYSDIR>config.

Keywords and command line formats

SSLCertificateFile <pem:filename> <der:filename>

```
-sslcf filename
```

This parameter is required. It identifies the server's Certificate file, which can optionally contain the server's Private Key file (contained in the same file, not recommended). If the Private Key is encrypted the pass phrase must be given (see SSLCertificateKeyPass). If the filename is not preceded with pem: or der: then pem format is assumed.

SSLCertificateKeyFile <pem:filename> <der:filename>

```
-sslckf filename
```

Identifies the server's Private Key file. If the server's Private Key file is not combined with the server's Certificate in the SSLCertificateFile, use this additional directive to point to the stand-alone Private Key file (recommended). If the Private Key is encrypted the pass phrase must be given (see SSLCertificateKeyPass).

SSLCertificateKeyPass <passphrase>

```
-sslckp phrase
```

Specifies the optional pass phrase for the server's encrypted private key in the SSLCertificateFile or SSLCertificateKeyFile.

SSLCaCertificateFile <filename>-sslca filename

Identifies a file where you can assemble the Certificates of Certification Authorities (CAs) whose clients you wish to be able to authenticate. The file is simply the concatenation of the various PEM-encoded Certificate files, in order of preference. This can be used alternatively and/or additionally to SSLCACertificatePath.

SSLCaCertificatePath <directory>-sslcap path

Identifies the directory where you keep the Certificates of Certification Authorities (CAs) whose clients you wish to be able to authenticate. The files in this directory have to be PEM-encoded and are accessed through hash filenames. You can't just place the Certificate files in this directory, you also have to create symbolic links named `hash-value.N` that point to the real files. This directory must contain the appropriate symbolic links.

SSLVerifyClient <level>-sslvc n

Sets the Certificate verification level for the Client Authentication. It is used in the client authentication process during the SSL handshake when a connection is established. The following values are available for level:

0. no client Certificate is required at all (default)
1. the client may present a valid Certificate
2. the client must present a valid Certificate
3. the client may present a valid Certificate but it needn't be verifiable

Level 0 means the server will ignore any certificates offered and will allow all connections.

Level 1 means that if the remote end presents no certificate, the server will accept the connection. If a certificate is presented, is valid and can be verified, the connection is accepted. If the certificate is invalid, drop the connection.

Level 2 means that the server requires and verifies certificates for every SSL connection. If no certificate, an invalid certificate or a certificate that cannot be verified is presented, then it will drop the connection.

Level 3 means that if the remote end offers no certificate, the connection is accepted. If it offers a certificate, it must be valid, but need not be verifiable with a CA in the certificate store.

SSLVerifyDepth <depth>-sslvd n

Sets how deeply GwebSS should verify before deciding that the clients don't have a valid certificate. The depth actually is the maximum number of intermediate certificate issuers.

0. means that only self-signed client certificates are accepted

1. (default) means the client certificate can be self-signed or has to be signed by a CA which is directly known to the server (i.e. the CA's certificate is under `SSLCACertificatePath`).

Level 0 means that only self-signed client certificates (which are stored locally as trusted) will be accepted. Level 1 is the more normal arrangement whereby the client must have a valid certificate signed by a locally stored trusted CA. If intermediate CAs are involved, the level must be set high enough to verify each intermediate CA in the chain of CAs that leads back to the trusted root CA that issued the first certificate in the chain.

SSLSessionCacheTimeout `<seconds>-sslto n`

Sets the timeout for the information stored in the global/inter-process SSL Session Cache and the OpenSSL internal memory cache. It can be set as low as necessary for testing, but should be set to higher values such as 300 or more in real life. The default is 300 (5 minutes). This means that the results of the initial key negotiation and authentication procedure will be remembered for this period of time, so that a new connection from the same client will not suffer the overhead of the handshake.

Stunnel as an SSL server

There are many SSL-tunneling products available. We have qualified and use Stunnel to provide secure Telnet services on our in-house UNIX and Linux test platforms for Glink SSL clients connecting over the Internet.

Functionality

Stunnel is a wrapper that allows you to encrypt arbitrary TCP connections using SSL. It listens for incoming connections on a specified port, i.e. the designated secure port for some service, and – if the SSL authentication is successful – reroutes the connections to the internal port used by the standard, i.e. insecure, port for a service such as Telnet.

Download and installation

Stunnel is based on OpenSSL, which is an SSL toolkit. Detailed instructions for compiling and installing them are included in the respective source distributions. They can be found at

www.openssl.org

www.stunnel.org



Certificates

For a really good introduction to SSL and certificates see:

- o <http://www.ultranet.com/~fhirsch/Papers/wwwj/article.html>

X509 certificate content

Subject	Distinguished Name, Public Key
Issuer	Distinguished Name, Signature
Period of Validity	Not Before Date, Not After Date
Administrative Information	Version, Serial Number
Extended Information	

Distinguished name fields

Field	Abb.	Description	Example
Common Name	CN	Name being certified	CN=Frederick Hirsch
Organization	O	Name is associated with this organization	O=The Open Group
Organizational Unit	OU	Name is associated with this organization unit (department)	OU=Research Institute
Locality	L	Name is located in this City	L=Cambridge
State/Province	SP	Name is located in this State/Province	SP=Massachusetts
Country	C	Name is located in this Country	C=US (ISO code)



Certificates supplied by G&R

G&R supply the following certificates for test and demonstration purposes:

ca.crt	The issuer (CA - certificate authority) certificate.
glink.p12	Glink demo certificate and key in P12 format
ggate.pem	Ggate demo certificate and key in PEM file
gwebs.pem	Gweb demo certificate and key in PEM file

The supplied Ggate/Gweb certificates are generic certificates. When clients try to authenticate servers they usually look in the “Common Name” (CN) field in the certificate to see if it matches the DNS (Internet) name of the server.

On request G&R can supply you with server certificates that contain a server name of choice, making it possible for the client to correctly identify your server. Note that this certificate will still not be secure and should not be used in a production environment.

Getting started with G&R certificates

Ggate

To start *Ggate/ssl* using **ggate.pem** as the server certificate use:

```
gg_tcp -sslcf ggate.pem
```

This will start *Ggate* in SSL mode using `ggate.pem` as a server certificate.

Gweb

To start *Gweb/ssl* using **gweb.pem** as the server certificate use:

```
gwebss -sslcf gwebs.pem
```

or add the parameter 'SSLCertificateFile gwebs.pem' to `gwebs.XXX`.

Authentication of valid clients only

If you start Ggate or Gwebss with SSL as described above, it ensures that the Clients (Glink/Browser) are able to check that they are talking to the correct host, and that all traffic between them is encrypted.

If you want to secure the server further, so that only connections from valid clients are accepted, you can use the `-sslca` parameter. The file specified contains one or more issuer certificates. When this parameter is in use the server will ask the client for a certificate during connect, and will allow only clients with valid certificates issued by issuers found in the file specified. e.g.

```
gg_tcp -sslcf ggate.pem -sslca ca.crt
```

will only allow connections from clients with certificates issued by the G&R demo CA (`ca.crt`).

Glink

Glink uses certificates present in the standard Windows certificate store. The supplied ***glink.p12*** certificate can be imported into windows by either double-clicking it, or from Internet Explorer using Tools->Internet Options->Content->Certificates. This file is password protected and the password is ***glink***.

Trusted third parties

Server certificates that are intended to be secure should be ordered from a commercial CA. A list of available CAs worldwide can be found at:

<http://www.gmw.ac.uk/~t16345/ca.htm>

Among the most popular ones are:

- Thawte (www.thawte.com)
- Verisign (www.verisign.com)
- Geotrust (www.geotrust.com)
- Baltimore (www.baltimore.com)
- Entrust (www.entrust.com)
- GlobalSign (www.globalsign.com)
- IpsCA (www.ipsca.com)
- Comodo (www.comodogroup.com)
- FreeSSL (www.freessl.com)

In order to obtain a certificate from any of these CAs you must create a valid certificate request (CSR). There are several ways to do this. See below for an explanation of using the XCA program to create the request.

Creating your own certificates

In order to issue your own certificates you must obtain the necessary tools to set up a Certificate Authority and issue new certificates using this CA. These certificates can be considered secure within an organization. Unfortunately there are currently not very many good tools for certificate maintenance and those available are often very limited or very complex.

You can create your own client certificates using OpenSSL. OpenSSL comes with utilities and instructions on how to set up your own CA (Certification Authority) in order to create and sign your own certificates. A useful tool for creation of certificates is 'ssl.ca'. It can be found at:

<http://www.openssl.org/contrib/>

We have been using XCA (<http://www.hohnstaedt.de/xca.html>) to create the certificates. This is a windows based tool, which is configurable and relatively easy to use. It lets you set up one or more Certification Authorities, and create new server or client certificates using these CAs.

Using XCA to create a certificate request

- Select the "Certificate signing request" tab
- Click the "New Request" button
- The "Certificate request Wizard" is opened, click "Next"
- Use "Empty Template" and click "Next"
- Fill in all the necessary fields as you want them to appear in the certificate. Remember to set the "Common name" to the correct DNS name for your server (e.g. www.gar.no).
- Click the button "Generate a new key" to create a new private/public key pair for this request/certificate.
- Click "Next" and then "Finish". The certificate request is now created.
- Then click "Export" to export the request to a file.
- You now have a valid CSR file that can be used in order to obtain a valid certificate from a Certification Authority.
- Go to the "RSA Keys" tab, select the private key corresponding to the request and export it to a PEM file.
- The certificate received from your CA can be combined with the private key for use with GwebS or Ggate.

Request a test certificate from Thawte

In order to test the CSR created you can request a test certificate from Thawte. Test certificates will be issued by an unknown (insecure) CA and will be valid for only 3 weeks, but they are instantly available.

- Go to the Thawte web page at <http://www.thawte.com/buy.html>.
- Select one of the buttons labeled “try” under “Web Server Certificates”.
- Fill in the form as requested and click “submit” and then “next”.
- In the text box paste the content of the CSR file exported from XCA.
- Click next in order to download your new certificate.
- This certificate combined with its private key already generated with the request, can be used as a server certificate for GwebS or Ggate.
- If this test certificate works as expected you can order a working certificate from any of the listed CA’s above, using the same certificate request.



OpenSSL license details

Copyright (c) 1998-2003 The OpenSSL Project. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software must display the following acknowledgment:
"This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit. (<http://www.openssl.org/>)"
4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact openssl-core@openssl.org.
5. Products derived from this software may not be called "OpenSSL" nor may "OpenSSL" appear in their names without prior written permission of the OpenSSL Project.
6. Redistributions of any form whatsoever must retain the following acknowledgment:
"This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (<http://www.openssl.org/>)"

THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE OpenSSL PROJECT OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This product includes cryptographic software written by Eric Young (ey@cryptsoft.com). This product includes software written by Tim Hudson (tjh@cryptsoft.com).

Original SSLeay license

Copyright (C) 1995-1998 Eric Young (ey@cryptsoft.com) All rights reserved.

This package is an SSL implementation written by Eric Young (ey@cryptsoft.com). The

implementation was written so as to conform with Netscape's SSL.

This library is free for commercial and non-commercial use as long as the following conditions are adhered to. The following conditions apply to all code found in this distribution, be it the RC4, RSA, lhash, DES, etc., code; not just the SSL code. The SSL documentation included with this distribution is covered by the same copyright terms except that the holder is Tim Hudson (tjh@cryptsoft.com).

Copyright remains Eric Young's, and as such any Copyright notices in the code are not to be removed. If this package is used in a product, Eric Young should be given attribution as the author of the parts of the library used. This can be in the form of a textual message at program startup or in documentation (online or textual) provided with the package.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software must display the following acknowledgement:

"This product includes cryptographic software written by Eric Young (ey@cryptsoft.com)"

The word 'cryptographic' can be left out if the routines from the library being used are not cryptographic related :-).

4. If you include any Windows specific code (or a derivative thereof) from the apps directory (application code) you must include an acknowledgement:

"This product includes software written by Tim Hudson (tjh@cryptsoft.com)"

THIS SOFTWARE IS PROVIDED BY ERIC YOUNG ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The license and distribution terms for any publicly available version or derivative of this code cannot be changed. i.e. this code cannot simply be copied and put under another distribution license [including the GNU Public Licence.]