

Integrating Stunnel and OpenSSL with Teamware Office

Integrating Stunnel and OpenSSL with Teamware Office

June 2001
Document Version: 1.0

This document was produced by TeamWARE Group Oy in Helsinki, Finland.
We welcome and consider all comments and suggestions.

Please send them to:
TeamWARE Group Oy
P.O. Box 135
FIN-00381 Helsinki
Finland
Fax number: +358 9 5128 2705
Internet address:
office.team@teamware.com

Teamware Group endeavors to ensure that the information in this document is correct, but accepts no liability for any error or omission in the same.

The development of Teamware Group products and services is continuous and published information may not be up to date. Any particular issue of a product may contain part only of the facilities described in this document or may contain facilities not described herein. It is important to check the current position with Teamware Group.

Specifications and statements as to performance in this document are Teamware Group estimates intended for general guidance. They may require adjustments in particular circumstances and should therefore not be taken as formal offers or commitments.

This document is not part of a contract or license save insofar as may be expressly agreed.

Further information can primarily be obtained through Teamware partners. Contact you local Teamware contact for further details. Teamware can also offer support and other services on a consultancy fare basis. Alternatively support for answering questions of technical nature can be handled through Teamware Incident Database when agreement between the parties exist.

Teamware is a registered trademark in the USA and other countries. Microsoft, MS-DOS and Windows are registered trademarks of Microsoft Corporation in the USA and other countries. UNIX is a registered trademark of The Open Group. Netscape Navigator is a trademark of Netscape Communications Corporation in the USA and other countries. Mozilla is a trademark of Mozilla Organization in the USA and other countries. OpenSSL is a trademark of OpenSSL Project in the USA and other countries.

Disclaimer, warranty and terms of distribution

These publications are partly based on and contains pre-release or other vendor software that may vary at the level of performance, functionality or compatibility. The purpose of the publications is to give the Teamware users the opportunity to take a look of what can be done and explore these features. Teamware Group shall have no obligation to maintain, correct, update, change, modify or otherwise support the software components and documents mentioned in this publication.

Teamware Group expressly disclaims any warranty for the publication. The publication is provided "as is" without warranty of any kind, including, without limitation, the implied warranties of fitness for a particular purpose. In no event shall Teamware Group be liable for any damages whatsoever including, damages for loss of business profit, business interruption, loss of business information, or any other loss, occurring during the use of, or inability to use the software.

Important legal note: Please note that export/import of cryptography technologies, use of strong cryptography, distribution of technical details about cryptography technologies and/or establishing self-singed Certificates Authorities is regulated by specific laws in certain countries. Teamware Group cannot be held responsible in any way for any violation of such laws resulting from using this document and the information it contains.

Contents

| | |
|---|-----------|
| 1 Introduction | 5 |
| 1.1 About Stunnel (from the Stunnel documentation)..... | 5 |
| 1.2 About OpenSSL (from the OpenSSL documentation) | 6 |
| 2 About SSL | 8 |
| 2.1 What are SSL and TLS? | 8 |
| 2.2 The SSL protocol | 8 |
| 2.3 Ciphers used with SSL..... | 10 |
| 2.3.1 Cipher Suites with RSA Key Exchange..... | 11 |
| 2.3.2 FORTEZZA Cipher Suites | 12 |
| 2.4 The SSL Handshake..... | 12 |
| 2.5 Server Authentication | 14 |
| 2.5.1 Man in the middle attack..... | 16 |
| 2.6 Client authentication | 16 |
| 3 Installing OpenSSL | 18 |
| 4 Installing Stunnel..... | 19 |
| 5 Certificate management..... | 20 |
| 5.1 What certificates are needed? | 21 |
| 5.2 The Certificate Authority (CA)..... | 21 |
| 5.2.1 Being your own CA | 21 |
| 5.2.2 Using third party CAs | 23 |
| 5.3 The server certificate | 24 |
| 5.3.1 Create the server Certificate Signing Request | 24 |
| 5.3.2 Self-signing the server Certificate Signing Request | 25 |
| 5.3.3 Signing the server Certificate Signing Request by 3 rd party CAs | 27 |
| 5.4 Client certificates | 29 |
| 5.4.1 Creating the client Certificate Signing Request | 30 |
| 5.4.2 Self-signing the client Certificate Signing Request | 31 |
| 5.4.3 Signing the client Certificate Signing Request by 3 rd party CAs | 32 |
| 5.4.4 Exporting the client certificate | 32 |
| 5.4.4.1 Exporting client certificates in PKCS#12 (PFX) format | 32 |
| 5.4.4.2 Exporting client certificates in DER format | 33 |
| 5.4.5 Converting client certificates from PKCS#12 to PEM | 34 |
| 5.5 Creating private keys. | 35 |
| 6 Using Stunnel | 37 |

| | | |
|----------|--|-----------|
| 6.1 | Integrating Stunnel with Teamware Office services..... | 37 |
| 6.2 | Providing both SSL and non-SSL services..... | 39 |
| 6.3 | Automatically starting Stunnel | 39 |
| 6.4 | Running Stunnel on other machines than Teamware Office | 40 |
| 6.5 | Specifying the ciphers to be used by Stunnel | 41 |
| 6.6 | Restricting access using client security certificates | 42 |
| 6.6.1 | Server-side configurations for client certificates | 42 |
| 6.6.2 | Client-side configurations for client certificates | 44 |
| 6.6.2.1 | Using client certificates with Internet Explorer..... | 44 |
| 6.6.2.2 | Using client certificates with Netscape | 45 |
| 6.6.2.3 | Using client certificates with Mozilla..... | 46 |
| 6.7 | Integrating Stunnel with the Teamware Office Web Service | 47 |
| 6.8 | Using Stunnel with the Teamware Office MIME Connector..... | 48 |
| 7 | Additional resources..... | 50 |

1 Introduction

This document presents the techniques for integrating Stunnel and its underlying OpenSSL tools with Teamware Office.

Stunnel with OpenSSL can be used to provide SSL encryption for various Teamware Office services: HTTP, IMAP, POP3, SMTP and LDAP.

OpenSSL will provide the actual encryption mechanism for Stunnel through its libraries. Additionally, the OpenSSL tools can be used for generating and managing the security certificates.

The document will focus on two main issues:

- **Certificate management (Chapter 5)** - will present the methods for creating, signing and using different types of certificates (CA, server and client certificates) with the OpenSSL tools.
- **Using Stunnel with Teamware Office (Chapter 6)** - will present the actual methods for providing SSL encryption for Teamware Office services by using Stunnel.

It should not be considered as a complete manual for Stunnel or OpenSSL. You are encouraged to consult the Stunnel and OpenSSL specific documentation, available in the installation packages or on the products' Web sites: www.stunnel.org and www.openssl.org. Also, this document can't cover all the implementation possibilities and the reader is encouraged to look at this document as a starting point for finding more methods of integrating Stunnel and OpenSSL with Teamware Office environments.

1.1 About Stunnel (from the Stunnel documentation)

Stunnel does not contain any cryptographic code itself - instead it relies on external SSL libraries. It works with both [OpenSSL](http://www.openssl.org) and its precursor [SSLey](http://www.ssl.com). Both of these packages are capable of strong (128 bit) cryptography, and Stunnel will negotiate SSL connections of the highest strength available between client and server.

Stunnel can work by either:

- Receiving unencrypted data and sending it to an SSL server
- Receiving encrypted data and

- Sending the decrypted data to an arbitrary port on that or another machine
- Launching a local program to talk to the remote machine over the encrypted channel.

Stunnel has support for:

- Being an SSL client
- Being an SSL server
- Server and client side certificate verification
- TCP wrapper support
- IDENT lookups
- SMTP protocol negotiation
- Source address rewriting (transparency) (where supported by the OS)
- Restricting allowed SSL ciphers

Stunnel can help:

- Protect interception of data
- Prevent manipulation of data
- And, if compiled with libwrap support:
 - Defend against IP source routing, (one host sending packets as if they came from somewhere else)
 - DNS spoofing (an attacker forging name server records)

1.2 About OpenSSL (from the OpenSSL documentation)

OpenSSL is based on the excellent SSLeay library developed from Eric A. Young and Tim J. Hudson. The OpenSSL toolkit is licensed under a dual-license (the OpenSSL license plus the SSLeay license) situation, which basically means that you are free to get and use it for commercial and non-commercial purposes as long as you fulfill the conditions of both licenses.

Available ciphers:

- libdes – EAY's libdes DES encryption package which has been floating around the net for a few years. It includes 15 'modes/variations' of DES (1, 2 and 3 key versions of ecb, cbc, cfb and ofb; pcfc and a more general form of cfb and ofb) including desx in cbc mode, a fast crypt(3), and routines to read passwords from the keyboard.
- RC4 encryption
- RC2 encryption – 4 different modes, ecb, cbc, cfb and ofb.
- Blowfish encryption – 4 different modes, ecb, cbc, cfb and ofb.
- IDEA encryption – 4 different modes, ecb, cbc, cfb and ofb.

Available digests:

- MD5 and MD2 message digest algorithms, fast implementations,
- SHA (SHA-0) and SHA-1 message digest algorithms,
- MDC2 message digest. A DES based hash that is popular on smart cards.

Public keys used:

- RSA encryption/decryption/generation.
- DSA encryption/decryption/generation.
- Diffie-Hellman key-exchange/key generation.

X.509v3 certificates:

- X509 encoding/decoding into/from binary ASN1 and a PEM based ascii-binary encoding which supports encryption with a private key. Program to generate RSA and DSA certificate requests and to generate RSA and DSA certificates.

Systems:

The normal digital envelope routines and base64 encoding. Higher level access to ciphers and digests by name. New ciphers can be loaded at run time. The BIO io system which is a simple non-blocking IO abstraction. Current methods supported are file descriptors, sockets, socket accept, socket connect, memory buffer, buffering, SSL client/server, file pointer, encryption, digest, non-blocking testing and null.

The `openssl` command -A command line tool that can be used for:

- Creation of RSA, DH and DSA key parameters
- Creation of X.509 certificates, CSRs and CRLs
- Calculation of Message Digests
- Encryption and Decryption with Ciphers
- SSL/TLS Client and Server Tests
- Handling of S/MIME signed or encrypted mail

2 About SSL

Netscape, as developer of the SSL protocol, maintains a very good set of SSL documentation at: <http://developer.netscape.com> and you are encouraged to consult it if you want to learn more about SSL.

2.1 What are SSL and TLS?

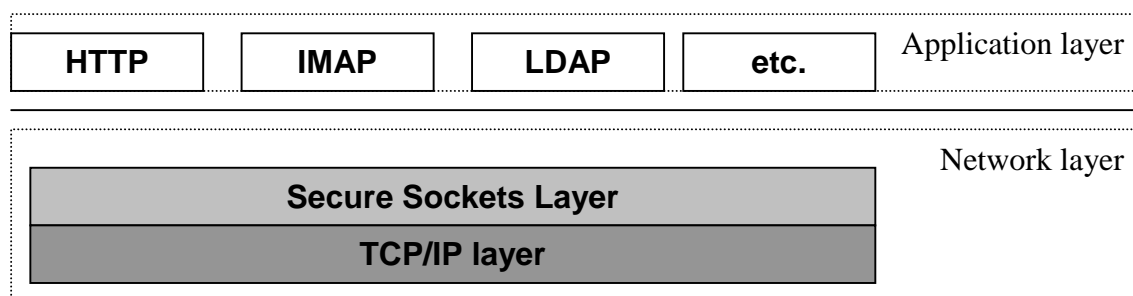
Secure Sockets Layer (SSL) is a protocol designed for securing electronic data transmissions between a client and a server. It provides protection against eavesdropping, tampering and forgery.

The protocol was originally developed by Netscape. Now all major web server vendors support SSL and TLS. Since its release in March 1996, the SSL protocol has become a standard for securing electronic transmissions across the Internet. To date Netscape's SSL developers, have been committed to keeping it an open, non-proprietary standard and have made the entire protocol available to the public.

The Internet Engineering Task Force (IETF) standard called Transport Layer Security (TLS) is based on SSL. They have recently published The TLS Protocol Version 1.0-Internet Draft, available online at <http://www.ietf.org/rfc/rfc2246.txt?number=2246>.

2.2 The SSL protocol

The Transmission Control Protocol/Internet Protocol (TCP/IP) governs the transport and routing of data over the Internet. Other protocols, such as the HyperText Transport Protocol (HTTP), Lightweight Directory Access Protocol (LDAP), or Internet Messaging Access Protocol (IMAP), run on top of TCP/IP in the sense that they all use TCP/IP to support typical application tasks such as displaying web pages or running email servers.



The SSL protocol runs above TCP/IP and below higher-level protocols such as HTTP or IMAP. It uses TCP/IP on behalf of the higher-level protocols, and in the process allows an SSL-enabled server to authenticate itself to an SSL-enabled client, allows the client to authenticate itself to the server, and allows both machines to establish an encrypted connection.

These capabilities address fundamental concerns about communication over the Internet and other TCP/IP networks:

- **SSL server authentication** allows a user to confirm a server's identity. SSL-enabled client software can use standard techniques of public-key cryptography to check that a server's certificate and public ID are valid and have been issued by a certificate authority (CA) listed in the client's list of trusted CAs. This confirmation might be important if the user, for example, is sending a credit card number over the network and wants to check the receiving server's identity.
- **SSL client authentication** allows a server to confirm a user's identity. Using the same techniques as those used for server authentication, SSL-enabled server software can check that a client's certificate and public ID are valid and have been issued by a certificate authority (CA) listed in the server's list of trusted CAs. This confirmation might be important if the server, for example, is a bank sending confidential financial information to a customer and wants to check the recipient's identity.
- **An encrypted SSL connection** requires all information sent between a client and a server to be encrypted by the sending software and decrypted by the receiving software, thus providing a high degree of confidentiality. Confidentiality is important for both parties to any private transaction. In addition, all data sent over an encrypted SSL connection is protected with a mechanism for detecting tampering--that is, for automatically determining whether the data has been altered in transit.

The SSL protocol includes two sub-protocols: the SSL record protocol and the SSL handshake protocol. The SSL record protocol defines the format used to transmit data. The SSL handshake protocol involves using the SSL record protocol to exchange a series of messages between an SSL-enabled server and an SSL-enabled client when they first establish an SSL connection. This exchange of messages is designed to facilitate the following actions:

- Authenticate the server to the client.
- Allow the client and server to select the cryptographic algorithms, or ciphers, that they both support.
- Optionally authenticate the client to the server.
- Use public-key encryption techniques to generate shared secrets.
- Establish an encrypted SSL connection.

2.3 Ciphers used with SSL

The SSL protocol supports the use of a variety of different cryptographic algorithms, or **ciphers**, for use in operations such as authenticating the server and client to each other, transmitting certificates, and establishing session keys. Clients and servers may support different **cipher suites**, or sets of ciphers, depending on factors such as the version of SSL they support, company policies regarding acceptable encryption strength, and government restrictions on export of SSL-enabled software.

Among its other functions, the SSL handshake protocol determines how the server and client negotiate which cipher suites they will use to authenticate each other, to transmit certificates, and to establish session keys.

The list below presents the ciphers supported by the SSL protocol. Although the majority of these ciphers are supported by OpenSSL, it is recommended to check the release notes for the OpenSSL version that you are using.

- **DES.** Data Encryption Standard, an encryption algorithm used by the U.S. Government.
- **DSA.** Digital Signature Algorithm, part of the digital authentication standard used by the U.S. Government.
- **KEA.** Key Exchange Algorithm, an algorithm used for key exchange by the U.S. Government.
- **MD5.** Message Digest algorithm developed by Rivest.
- **RC2 and RC4.** Rivest encryption ciphers developed for RSA Data Security.
- **RSA.** A public-key algorithm for both encryption and authentication. Developed by Rivest, Shamir, and Adleman.
- **RSA key exchange.** A key-exchange algorithm for SSL based on the RSA algorithm.
- **SHA-1.** Secure Hash Algorithm, a hash function used by the U.S. Government.
- **SKIPJACK.** A classified symmetric-key algorithm implemented in FORTEZZA-compliant hardware used by the U.S. Government.
- **Triple-DES.** DES applied three times.

Key-exchange algorithms like KEA and RSA key exchange govern the way in which the server and client determine the symmetric keys they will both use during an SSL session. The most commonly used SSL cipher suites use RSA key exchange.

The SSL 2.0 and SSL 3.0 protocols support overlapping sets of cipher suites. Administrators can enable or disable any of the supported cipher suites for both clients and servers. When a particular client and server exchange information during the SSL handshake, they identify the strongest enabled cipher suites they have in common and use those for the SSL session.

Decisions about which cipher suites a particular organization decides to enable depend on trade-offs among the sensitivity of the data involved, the speed of the cipher, and the applicability of export rules.

2.3.1 Cipher Suites with RSA Key Exchange

The list below lists the cipher suites supported by SSL that use the RSA key-exchange algorithm. Unless otherwise indicated, all ciphers listed in the table are supported by both SSL 2.0 and SSL 3.0. Cipher suites are listed from strongest to weakest.

- **Triple DES, which supports 168-bit encryption, with SHA-1 message authentication.** Triple DES is the strongest cipher supported by SSL, but it is not as fast as RC4. Triple DES uses a key three times as long as the key for standard DES. Because the key size is so large, there are more possible keys than for any other cipher--approximately $3.7 * 10^{50}$. Both SSL 2.0 and SSL 3.0 support this cipher suite.
- **RC4 with 128-bit encryption and MD5 message authentication.** Because the RC4 and RC2 ciphers have 128-bit encryption, they are the second strongest next to Triple DES (Data Encryption Standard), with 168-bit encryption. RC4 and RC2 128-bit encryption permits approximately $3.4 * 10^{38}$ possible keys, making them very difficult to crack. RC4 ciphers are the fastest of the supported ciphers. Both SSL 2.0 and SSL 3.0 support this cipher suite.
- **RC2 with 128-bit encryption and MD5 message authentication.** Because the RC4 and RC2 ciphers have 128-bit encryption, they are the second strongest next to Triple DES (Data Encryption Standard), with 168-bit encryption. RC4 and RC2 128-bit encryption permits approximately $3.4 * 10^{38}$ possible keys, making them very difficult to crack. RC2 ciphers are slower than RC4 ciphers. This cipher suite is supported by SSL 2.0 but not by SSL 3.0.
- **DES, which supports 56-bit encryption, with SHA-1 message authentication.** DES is stronger than 40-bit encryption, but not as strong as 128-bit encryption. DES 56-bit encryption permits approximately $7.2 * 10^{16}$ possible keys. Both SSL 2.0 and SSL 3.0 support this cipher suite, except that SSL 2.0 uses MD5 rather than SHA-1 for message authentication.
- **RC4 with 40-bit encryption and MD5 message authentication.** RC4 40-bit encryption permits approximately $1.1 * 10^{12}$ (a trillion) possible keys. RC4 ciphers are the fastest of the supported ciphers. Both SSL 2.0 and SSL 3.0 support this cipher.
- **RC2 with 40-bit encryption and MD5 message authentication.** RC2 40-bit encryption permits approximately $1.1 * 10^{12}$ (a trillion) possible keys. RC2 ciphers are slower than the RC4 ciphers. Both SSL 2.0 and SSL 3.0 support this cipher.
- **No encryption, MD5 message authentication only.** This cipher suite uses MD5 message authentication to detect tampering. It is typically supported in case a client and server have none of the other ciphers in common. This cipher suite is supported by SSL 3.0 but not by SSL 2.0.

Note that for RC4 and RC2 ciphers, the phrase "40-bit encryption" means the keys are still 128 bits long, but only 40 bits have cryptographic significance.

2.3.2 FORTEZZA Cipher Suites

The list below presents additional cipher suites supported by Netscape products with FORTEZZA for SSL 3.0. FORTEZZA is an encryption system used by U.S. government agencies to manage sensitive but unclassified information. It provides a hardware implementation of two classified ciphers developed by the federal government: FORTEZZA KEA and SKIPJACK. FORTEZZA ciphers for SSL use the Key Exchange Algorithm (KEA) instead of the RSA key-exchange algorithm mentioned in the preceding section, and use FORTEZZA cards and DSA for client authentication.

- **RC4 with 128-bit encryption and SHA-1 message authentication.** Like RC4 with 128-bit encryption and MD5 message authentication, this cipher is one of the second strongest ciphers after Triple DES. It permits approximately $3.4 * 10^{38}$ possible keys, making it very difficult to crack. This cipher suite is supported by SSL 3.0 but not by SSL 2.0.
- **RC4 with SKIPJACK 80-bit encryption and SHA-1 message authentication.** The SKIPJACK cipher is a classified symmetric-key cryptographic algorithm implemented in FORTEZZA-compliant hardware. Some SKIPJACK implementations support key escrow using the Law Enforcement Access Field (LEAF). The most recent implementations do not. This cipher suite is supported by SSL 3.0 but not by SSL 2.0.
- **No encryption, SHA-1 message authentication only.** This cipher uses SHA-1 message authentication to detect tampering. This cipher suite is supported by SSL 3.0 but not by SSL 2.0.

2.4 The SSL Handshake

The SSL protocol uses a combination of public-key and symmetric key encryption. Symmetric key encryption is much faster than public-key encryption, but public-key encryption provides better authentication techniques.

An SSL session always begins with an exchange of messages called the **SSL handshake**. The handshake allows the server to authenticate itself to the client using public-key techniques, then allows the client and the server to cooperate in the creation of symmetric keys used for rapid encryption, decryption, and tamper detection during the session that follows. Optionally, the handshake also allows the client to authenticate itself to the server.

The steps involved can be summarized as follows:

1. The client sends the server the client's SSL version number, cipher settings, randomly generated data, and other information the server needs to communicate with the client using SSL.
2. The server sends the client the server's SSL version number, cipher settings, randomly generated data, and other information the client needs to communicate with the server over SSL. The server also sends its own certificate and, if the client is requesting a server resource that requires client authentication, requests the client's certificate.
3. The client uses some of the information sent by the server to authenticate the server. If the server cannot be authenticated, the user is warned of the problem and informed that an encrypted and authenticated connection cannot be established. If the server can be successfully authenticated, the client goes on to Step 4.
4. Using all data generated in the handshake so far, the client (with the cooperation of the server, depending on the cipher being used) creates the **premaster secret** for the session, encrypts it with the server's public key (obtained from the server's certificate, sent in Step2), and sends the encrypted premaster secret to the server.
5. If the server has requested client authentication (an optional step in the handshake), the client also signs another piece of data that is unique to this handshake and known by both the client and server. In this case the client sends both the signed data and the client's own certificate to the server along with the encrypted premaster secret.
6. If the server has requested client authentication, the server attempts to authenticate the client. If the client cannot be authenticated, the session is terminated. If the client can be successfully authenticated, the server uses its private key to decrypt the premaster secret, then performs a series of steps (which the client also performs, starting from the same premaster secret) to generate the **master secret**.
7. Both the client and the server use the master secret to generate the **session keys**, which are symmetric keys used to encrypt and decrypt information exchanged during the SSL session and to verify its integrity--that is, to detect any changes in the data between the time it was sent and the time it is received over the SSL connection.
8. The client sends a message to the server informing it that future messages from the client will be encrypted with the session key. It then sends a separate (encrypted) message indicating that the client portion of the handshake is finished.
9. The server sends a message to the client informing it that future messages from the server will be encrypted with the session key. It then sends a separate (encrypted) message indicating that the server portion of the handshake is finished.
10. The SSL handshake is now complete, and the SSL session has begun. The client and the server use the session keys to encrypt and decrypt the data they send to each other and to validate its integrity.

Before continuing with the session, servers can be configured to check that the client's certificate is present in the user's entry in an LDAP directory or in another storage space (e.g. a directory on the server's disk). This configuration option provides one way of ensuring that the client's certificate has not been revoked.

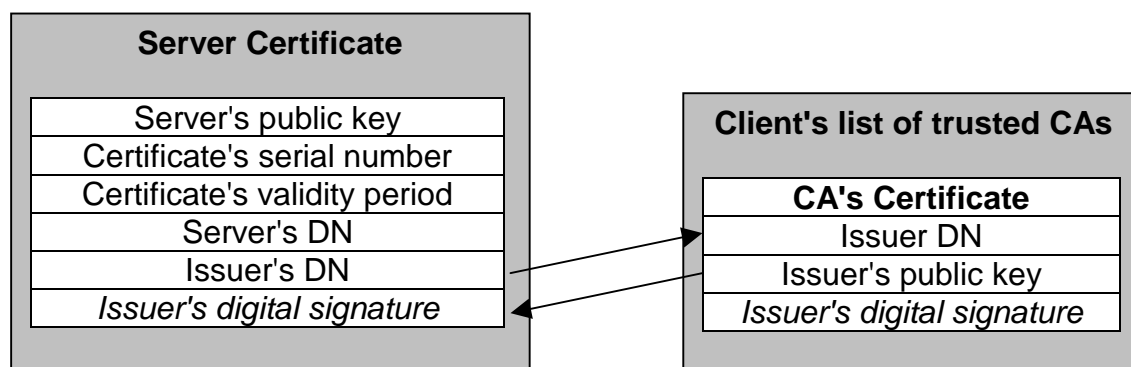
It's important to note that both client and server authentication involve encrypting some piece of data with one key of a public-private key pair and decrypting it with the other key:

- In the case of server authentication, the client encrypts the premaster secret with the server's public key. Only the corresponding private key can correctly decrypt the secret, so the client has some assurance that the identity associated with the public key is in fact the server with which the client is connected. Otherwise, the server cannot decrypt the premaster secret and cannot generate the symmetric keys required for the session, and the session will be terminated.
- In the case of client authentication, the client encrypts some random data with the client's private key--that is, it creates a digital signature. The public key in the client's certificate can correctly validate the digital signature only if the corresponding private key was used. Otherwise, the server cannot validate the digital signature and the session is terminated.

2.5 Server Authentication

The SSL-enabled client software always requires server authentication, or cryptographic validation by a client of the server's identity. As explained in Step 2 of the SSL Handshake, the server sends the client a certificate to authenticate itself. The client uses the certificate in Step 3 to authenticate the identity the certificate claims to represent.

To authenticate the binding between a public key and the server identified by the certificate that contains the public key, an SSL-enabled client must receive a "yes" answer to the four questions shown in the figure below. Although the fourth question is not technically part of the SSL protocol, it is the client's responsibility to support this requirement, which provides some assurance of the server's identity and thus helps protect against a form of security attack known as "man in the middle."



An SSL-enabled client goes through the following steps to authenticate a server's identity:

1. **Is today's date within the validity period?**
2. **Is the issuing CA a trusted CA?** Each SSL-enabled client maintains a list of trusted CA certificates. This list determines which server certificates the client will accept. If the distinguished name (DN) of the issuing CA matches the DN of a CA on the client's list of trusted CAs, the answer to this question is yes, and the client goes on to Step 3. If the issuing CA is not listed in the client's list of trusted CAs, depending on the client settings, the connection will either be dropped automatically or it will ask for the user's confirmation before proceeding to the Step 3. Additionally, especially in the case of Web browsers, the user might be presented with the option of adding this CA in the list of trusted CAs, either temporarily (for this SSL session only) or permanently.
3. **Does the issuing CA's public key validate the issuer's digital signature?** The client uses the public key from the CA's certificate (which it found in its list of trusted CAs in step 2) to validate the CA's digital signature on the server certificate being presented. If the information in the server certificate has changed since it was signed by the CA or if the CA certificate's public key doesn't correspond to the private key used by the CA to sign the server certificate, the client won't authenticate the server's identity. In certain cases, it might ask for a decision from the user. If the CA's digital signature can be validated, the server treats the user's certificate as a valid "letter of introduction" from that CA and proceeds. At this point, the client has determined that the server certificate is valid. It is the client's responsibility to take Step 4 before Step 5.
4. **Does the domain name in the server's certificate match the domain name of the server itself?** This step confirms that the server is actually located at the same network address specified by the domain name in the server certificate. Although step 4 is not technically part of the SSL protocol, it provides the only protection against a form of security attack known as Man in the middle attack. Clients must perform this step and must either refuse to authenticate the server or at least ask for user's decision before proceeding to Step 5.

5. **The server is authenticated.** The client proceeds with the SSL handshake. If the server requires client authentication, the server performs the steps described in 2.6 Client authentication.

After the steps described here, the server must successfully use its private key to decrypt the premaster secret the client sends during the SSL Handshake. Otherwise, the SSL session will be terminated. This provides additional assurance that the identity associated with the public key in the server's certificate is in fact the server with which the client is connected.

2.5.1 Man in the middle attack

As suggested in Step 4 above, the client application must check the server domain name specified in the server certificate against the actual domain name of the server with which the client is attempting to communicate. This step is necessary to protect against a man-in-the-middle attack, which works as follows.

The "man in the middle" is a rogue program that intercepts all communication between the client and a server with which the client is attempting to communicate via SSL. The rogue program intercepts the legitimate keys that are passed back and forth during the SSL handshake, substitutes its own, and makes it appear to the client that it is the server, and to the server that it is the client.

The encrypted information exchanged at the beginning of the SSL handshake is actually encrypted with the rogue program's public key or private key, rather than the client's or server's real keys. The rogue program ends up establishing one set of session keys for use with the real server, and a different set of session keys for use with the client. This allows the rogue program not only to read all the data that flows between the client and the real server, but also to change the data without being detected. Therefore, it is extremely important for the client to check that the domain name in the server certificate corresponds to the domain name of the server with which a client is attempting to communicate in addition to checking the validity of the server certificate.

2.6 Client authentication

SSL-enabled servers can be configured to require client authentication, or cryptographic validation by the server of the client's identity. When a server configured this way requests client authentication (see Step 6 of the SSL Handshake), the client sends the server both a certificate and a separate piece of digitally signed data to authenticate itself. The server uses the digitally signed data to validate the public key in the certificate and to authenticate the identity the certificate claims to represent.

The SSL protocol requires the client to create a digital signature by creating a one-way hash from data generated randomly during the handshake and known only to the client and server. The hash of the data is then encrypted with the private key that corresponds to the public key in the certificate being presented to the server.

To authenticate the binding between the public key and the person or other entity identified by the certificate that contains the public key, an SSL-enabled server must receive a "yes" answer to the first four questions presented below. Although the fifth question is not part of the SSL protocol, SSL servers can be configured to support this requirement to take advantage of the user's entry in an LDAP directory or other storage.

1. **Does the user's public key validate the user's digital signature?** The server checks that the user's digital signature can be validated with the public key in the certificate. If so, the server has established that the public key asserted to belong to the user matches the private key used to create the signature and that the data has not been tampered with since it was signed.

At this point, however, the binding between the public key and the DN specified in the certificate has not yet been established. The certificate might have been created by someone attempting to impersonate the user. To validate the binding between the public key and the DN, the server must also complete Step 3 and Step 4.

2. **Is today's date within the validity period?**
3. **Is the issuing CA a trusted CA or is the user's certificate in the list of trusted certificates?** An SSL-enabled server maintains a list of trusted CA certificates and/or a list of trusted client certificates.
4. **Does the issuing CA's public key validate the issuer's digital signature (if list of trusted CAs is used)?** The server uses the public key from the CA's certificate (which it found in its list of trusted CAs) to validate the CA's digital signature on the certificate being presented. If the CA's digital signature can be validated, the server treats the user's certificate as a valid "letter of introduction" from that CA. At this point, the SSL protocol allows the server to consider the client authenticated.
5. **(Optional) Is the user's certificate listed as valid for the user?** This optional step provides one way for a system administrator to reject a user's certificate even if it passes the tests in all the other steps. The server can maintain a list of valid certificates for each user, using an LDAP server or other storage.
6. **(Optional) Is the authenticated client authorized to access the requested resources?** The server checks what resources the client is permitted to access according to the server's access control lists (ACLs) and establishes a connection with appropriate access.

3 Installing OpenSSL

The OpenSSL source code can be obtained from a number of Web sites, starting with www.openssl.org. The downloadable packages and the OpenSSL site contain detailed compilation information for all the relevant platforms and therefore this issue will not be covered in the present document.

OpenSSL precompiled binaries are included in certain Linux distributions, so, in case you will be using Linux, you should check your specific Linux distribution for OpenSSL information.

At the time of writing this document, the latest stable version of OpenSSL was 0.9.6. You are encouraged to check constantly the OpenSSL related sites for new releases and updates.

For UNIX and Linux systems you will need to install OpenSSL in order to use Stunnel and to be able to generate and manage security certificates.

For Windows systems it is possible to run Stunnel without installing the entire OpenSSL package. Please see the next section for more details.

4 Installing Stunnel

The Stunnel source code can be obtained from various Internet sites, starting with www.stunnel.org. The download packages and the Stunnel Web site contain compilation and installation information for various platforms and therefore this issue is not covered in this document.

In order to be able to use Stunnel, you will need the OpenSSL libraries. These can be obtained by compiling and installing the OpenSSL package or, for Windows systems, by downloading the pre-compiled OpenSSL DLLs (see below for details).

Stunnel precompiled binaries are included in certain Linux distributions, so, in case you will be using Linux, you should check your specific Linux distribution for OpenSSL information.

Windows precompiled binaries are also available from the Stunnel Web site. Additionally, precompiled OpenSSL libraries are available (`libeay32.dll` and `libssl32.dll`). These libraries will provide the basic OpenSSL functionality that will enable Stunnel to run, but you will not be able to use them for certificate generation and for other advanced functions, you will need the complete OpenSSL package for that.

Also from the Stunnel Web site you can download a demo server certificate (`stunnel.pem`), which you can use for testing Stunnel without needing to go through the process of generating or obtaining your own certificates.

5 Certificate management

Before starting to generate certificates, it is recommended to customize the OpenSSL configuration file, which is usually `openssl.cnf` in the OpenSSL installation directory. The most important parameters are:

| | |
|---------------------------|---|
| <code>dir</code> | Is the default working directory for OpenSSL. At minimum, you will use this directory for storing the certificates database (the <code>index.txt</code> file) and the serial numbers for certificates issues by you (the <code>serial</code> file). The default value is <code>./demoCA</code> . You will have to use the same directory each time you issue a new certificates, otherwise the certificate database will not be accurate. If you plan to run the <code>openssl</code> commands from different directories, it is better to declare the full path for this directory. In either case, you must create the directory prior to proceeding to certificate generation. |
| <code>nsCertType</code> | Declares what can the generated certificates be used for. For CA and server certificates, you must use the value <code>server</code> . Other possible values are <code>client</code> , <code>email</code> and <code>objsign</code> . You can declare more usage scopes for a certificate, but it is not advisable to generate certificates that can be used both for server and client authentication. |
| <code>default_bits</code> | The default number of bits to be used for certificate requests. Unless you have very good reasons for not doing so, you should set it to 1024. |
| <code>nsComment</code> | Sets a comment that will be placed in the certificates. For example, you could set it to "MyCompany own Certificate Authority". Not all clients are using this field, but some do, such as the Netscape and Mozilla browsers. |

There are others parameters that you might want to modify according to your needs, such as the default values for certain fields (`countryName_default`, `commonName_default` etc.). Please see the comments in the configuration file and the OpenSSL documentation for more details.

After setting up the `openssl.cnf` file, create a directory, for example `cr`, in the OpenSSL installation directory, to be used for storing your files.

Please note that the directories and file names presented in this chapter are only suggestions. You can use any names suitable for your system, as long as you use them in a consistent manner.

5.1 What certificates are needed?

In order to operate an SSL environment, the following certificate types are needed:

- **CA certificates** - They are a "proof of confidence" for your servers. If you intend to act as a self-signing Certificate Authority, you will need to create these certificates. The other option is to have your server and client certificates signed by a third party CA. Technically speaking, there is no difference between a self-signed certificate and a certificate signed by a third party CA. The choice depends on the level of confidence required by your clients.
- **Server certificates** - These are the certificates that will be actually used for identifying your server to the clients and for encrypting the data transmissions.
- **Client certificates** - Unlike the first two types of certificates, these are optional. They are needed only if you want to allow access to specific resources only for certain clients.

5.2 The Certificate Authority (CA)

5.2.1 Being your own CA

If you want to be your own CA, you will need to create a CA self-signed certificate. This can be accomplished by running the following command in the OpenSSL installation directory (all in one line):

```
openssl req -new -x509 -config ./openssl.cnf -newkey rsa:1024 -md5  
-keyout ./cr/CAkey.pem -out ./cr/CAcert.pem -days 365
```

The parameters have the following meaning:

| | |
|------------------------------------|---|
| <code>req</code> | The name of the certificate generation and management utility in the OpenSSL package. |
| <code>-new</code> | Generate a new certificate request. |
| <code>-x509</code> | Self sign the certificate request. |
| <code>-config ./openssl.cnf</code> | The OpenSSL configuration file to be used. |
| <code>-newkey rsa:1024</code> | The type (RSA) and length (1024) of the private key. |

- md5 The digest algorithm used for signing the certificate (MD5). Other options can be -sha1 (SHA1), -md2 (MD2), -mdc2 (MDC2)
- keyout ./cr/CAkey.pem The file for storing the created private key.
- out ./cr/CAcert.pem The file for storing the created certificate.
- days 365 The validity period of the certificate, in days.

The output of the command looks as follows. The lines in boldface will require some input from you:

```
Using configuration from ./openssl.cnf
Generating a 1024 bit RSA private key
.....++++++
.....++++++
writing new private key to './cr/CAkey.pem'
Enter PEM pass phrase:
Verifying password - Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be
incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a
DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [FI]:
State or Province Name (full name) [Helsinki]:
Locality Name (eg, city) [Helsinki]:
Organization Name (eg, company) [MyCompany]:
Organizational Unit Name (eg, section) [MyCompany HQ]:
Common Name (eg, your name or your server's hostname)
[www.company.com]:
Email Address [root@company.com]:
```

The PEM pass phrase is a password that you must remember, as you will need it every time you will use the newly generated private key, e.g. for generating server certificates.

The rest of the information you are asked for is used for composing the Distinguished Name (DN) for this certificate. In this example, the default values from the configuration files were used, but you can specify new values.

The e-mail address that you specify will be the address that the clients will see when using your certificates. You should specify the e-mail address of the person responsible with security issues in your organization.

This command will create your self-signed CA certificate `CAcert.pem` and the private key file `CAcert.pem`, which you can use later for creating client and server certificates.

IMPORTANT: keep the PEM pass phrase and the generated files in secure locations, otherwise anyone could generate fake certificates in your name. The files should be readable only by the system administrator. For example, on UNIX systems, the files should be owned by root and the permission mode should be set to 400.

5.2.2 Using third party CAs

If you intend to use certificates signed by third party CAs, you will not generate any CA certificates. Instead, you will use the tools or services provided by the CAs for signing your server and client certificates. You will have to check with the CA that you want to use for the exact procedures.

In the section 5.3.1 it is presented how you can generate Certificate Signing Requests, which could be used with third party CAs, unless they have some other procedures and tools for generating them.

5.3 The server certificate

5.3.1 Create the server Certificate Signing Request

Regardless of the fact that you will have your certificates signed by yourself or by a third party CA, you must first generate a Certificate Signing Request. This can be done by running in the OpenSSL installation directory the following command (all in one line):

```
openssl req -new -config ./openssl.cnf -newkey rsa:1024 -md5 -nodes  
-keyout ./cr/SRVkey.pem -out ./cr/SRVreq.pem -days 365
```

The parameters have the same meaning as it was presented in section **5.2.1 Being your own CA**.

Please note that the `-x509` parameter is not used anymore, since this is not a self-signed certificate, it is a Certificate Signing Request.

Please also note the new parameter `-nodes`, which means that the private key will not be encrypted and you will not be asked for a PEM pass phrase. You could omit this parameter, but in this case you will need to enter the PEM pass phrase each time you want to use the certificate you are generating. This is not very convenient if you want to use this server certificate for a server process that needs to start automatically (for example a Web server or a mail server).

A sample output of this command is presented below. The lines that require you to enter some information are printed in boldface.

```
Using configuration from ./openssl.cnf  
Generating a 1024 bit RSA private key  
.....++++++  
...++++++  
writing new private key to './cr/SRVkey.pem'  
-----  
You are about to be asked to enter information that will be  
incorporated  
into your certificate request.  
What you are about to enter is what is called a Distinguished Name or a  
DN.  
There are quite a few fields but you can leave some blank  
For some fields there will be a default value,  
If you enter '.', the field will be left blank.  
-----  
Country Name (2 letter code) [FI]:  
State or Province Name (full name) [Helsinki]:  
Locality Name (eg, city) [Helsinki]:  
Organization Name (eg, company) [MyCompany]:  
Organizational Unit Name (eg, section) [MyCompany HQ]:  
Common Name (eg, your name or your server's hostname)  
[www.company.com]:  
Email Address [root@company.com]:
```

Please enter the following 'extra' attributes
to be sent with your certificate request

A challenge password []:

An optional company name []:

Again, you will be asked for similar information as when generating the self-signed CA certificate (see section 5.2.1). Additionally, you are prompted for a challenge password and an optional company name. In most cases you can leave these fields empty, but certain signing CAs might require them.

IMPORTANT: keep the generated files in secure locations, otherwise anyone could generate fake certificates in your name. The files should be readable only by the system administrator. For example, on UNIX systems, the files should be owned by root and the permission mode should be set to 400.

5.3.2 Self-signing the server Certificate Signing Request

If you want to be your own CA, you can sign the Certificate Signing Request with your CA certificate.

For this you can run in the OpenSSL installation directory a command like the one below (everything in one line):

```
openssl ca -config ./openssl.cnf -policy policy_match -days 365 -md md5  
-out ./cr/SRVcert.pem -keyfile ./cr/CAkey.pem -cert ./cr/CAcert.pem  
-outdir ./cr -infiles ./cr/SRVreq.pem
```

Some parameters used here were already presented in section 5.2.1 **Being your own CA**. The new parameters are:

| | |
|--------------------------------------|--|
| <code>ca</code> | The OpenSSL CA application |
| <code>-policy policy_match</code> | Specifies the "policy" for determining if a match exists between the certificate and the CA information. The name <code>policy_match</code> specifies a the section <code>[policy_match]</code> from the OpenSSL configuration file. |
| <code>-md md5</code> | The type of digest to use. Instead of <code>md5</code> , you can also use <code>sha1</code> or <code>mdc2</code> . |
| <code>-keyfile ./cr/CAkey.pem</code> | Specifies the files containing the private key. This file was generated in section 5.2.1, when you created your CA. |
| <code>-cert ./cr/CAcert.pem</code> | The CA certificate file to use for signing the Certificate Signing Request. This file was |

generated in section 5.2.1, when you created your CA.

`-outdir ./cr` The directory where all output files will be placed.

`-infile ./cr/SRVreq.pem` This parameter specifies the file or files containing the Certificate Signing Requests that need to be signed. It must be the last parameter specified, because everything following it will be a considered to be file names.

A sample output of this command is presented below. The lines that requires some input from the user printed in boldface:

```
Using configuration from ./openssl.cnf
Enter PEM pass phrase:
Check that the request matches the signature
Signature ok
The Subjects Distinguished Name is as follows
countryName          :PRINTABLE:'FI'
stateOrProvinceName  :PRINTABLE:'Helsinki'
localityName         :PRINTABLE:'Helsinki'
organizationName     :PRINTABLE:'MyCompany'
organizationalUnitName:PRINTABLE:'MyCompany HQ'
commonName           :PRINTABLE:'www.company.com'
emailAddress         :IA5STRING:'root@company.com'
Certificate is to be certified until May 31 16:37:57 2002 GMT (365
days)
Sign the certificate? [y/n]:y

1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
```

The PEM phrase that you are asked for is the one that you set when you have generated the CA private key (see section 5.2.1 **Being your own CA**).

After you answer **y** when you are asked if you want to commit the action, the files `index.txt` and `serial`, which represent the database for storing the signed certificates, are updated, and backup copies are saved in `index.txt.old` and `serial.old`. If you want to repeat the certificate signing process, revert first to the backup copies of these files, and also delete the file `xx.pem` (located in the directory specified by `-outdir`), where `XX` is the number that was stored in the `serial` file prior to the certificate signing.

It is very important to use the same `index.txt` and `serial` files every time you generate new certificates, so you have accurate information about all the certificates that you signed. If not for anything else, you might need them in the future for revoking certificates using Certificate Revocation Lists (CRL).

The command will output the signed certificate in the file you specified (`SRVcert.pem`), and also in a file called `XX.pem`, where `XX` is the number that was stored in the `serial` file when you signed the certificate. You should move this file `XX.pem` in the directory `demoCA/newcerts` under the OpenSSL installation directory.

Then you must copy the content of the files `SRVkey.pem` and `SRVcert.pem` (exactly in this order) into a new file, for example `SRV.pem`. This is the file that you will be actually using as server certificate, because it contains the private key and the certificate itself.

Copy the `SRV.pem` file that you just created into the directory used by OpenSSL for storing the certificates that are used for SSL connection with the server (usually it is the directory `certs` under the OpenSSL installation directory; it is set by the `certs` parameter from the OpenSSL configuration file).

The final step is to run the command in this directory the `c_rehash` command, which will create a set of symbolic links to the certificate files. These links are used by OpenSSL to identify the certificates.

IMPORTANT: keep the generated files in secure locations, otherwise anyone could generate fake certificates in your name. The files should be readable only by the system administrator. For example, on UNIX systems, the files should be owned by root and the permission mode should be set to 400.

5.3.3 Signing the server Certificate Signing Request by 3rd party CAs

For getting your certificates signed by a third party CAs, you should consult those CAs for the specific procedures.

Unless the CA requires that the Certificate Signing Requests to be generated with its own tools, you should be able to use Certificate Signing Requests generated with the procedure presented in section 5.3.1.

For signing the certificate, the CA will ask you to supply the contents of the `SRVreq.pem` file that you created and then it will send you the signed certificate.

Place the signed certificate in a new file, for example `cr/3rdSRVcert.pem` (which is equivalent with the file `cr/SRVcert.pem` presented in section **5.3.2 Self-signing the server Certificate Signing Request**). Then follow the instruction presented at the end of section **5.3.2 Self-signing the server Certificate Signing Request** for creating the file that contains the private key and the certificate (you could use a name like `3rdSRV.pem` instead of `SRV.pem`) and for rehashing the `certs` directory.

It is useful to note that certain CAs, such as Thawte (www.thawte.com) offer free trials for certificate signing. To test your Certificate Signing Request, you can go to the address <https://www.thawte.com/cgi/server/test.exe> and paste the contents of the

SRVreq.pem file into the Certificate Signing Request box, then press on the "Generate Test Certificate" button on the bottom of the page.

If everything is OK, you should be presented with a page containing the signed certificate, which you should copy to a file like `cr/3rdSRVcert.pem`. Then follow the procedure presented above.

IMPORTANT: keep the generated files in secure locations, otherwise anyone could generate fake certificates in your name. The files should be readable only by the system administrator. For example, on UNIX systems, the files should be owned by root and the permission mode should be set to 400.

5.4 Client certificates

Client certificates are required if you want to allow access only for clients that have these client certificates.

The procedure for creating client certificates is quite similar with the procedure for creating server certificates, both for using own CA or a third party CA. The following points must be applied:

- The type for certificate, defined by the `nsCert Type` parameter in the OpenSSL configuration file should be `client`, `email` and/or `objsign`, depending on what do you want to allow the client certificate to be used for:
 - `client` - used for authenticating clients, such as Web browsers
 - `email` - used for signing and/or encrypting e-mail messages
 - `objsign` - used for object signing, for example signing applications

You can allow multiple usage types for a certificate, for example:

```
nsCert Type = client, email
```

- The Distinguished Names (DN) of all the certificates that you generate (client and server) must be unique. The recommend solution is to use different Common Names (CN) for each certificate that you generate and sign, keeping the rest of the DN common to all certificates. For example, if you plan to allow the same client certificate to be used by more people, you could choose a group name as CN for that certificate. Or, if you wish that each user in your system to have its own unique certificate, you can use the person's name or ID as CN.

In fact, OpenSSL will not allow you to sign a new certificate whose DN appears in the list of the certificates that you previously signed (they are stored in the `demoCA/index.txt` file mentioned in section **5.3.2 Self-signing the server Certificate Signing Request**). The same if you use a third party CA - you can't sign more certificates for the same DN, unless you delete the old ones first.

- After the certificate is signed (by your own CA or third party CA), you don't have to place it in the OpenSSL `certs` directory.
- Most client applications (such as Web browsers or e-mail clients) don't accept the PEM format for client certificates, so you will have to export them in PKCS#12 (also known as PFX) or DER format.
- In most cases, the SSL clients and/or servers won't allow access for a client certificate that doesn't have its validity period completely inside the validity period of the CA certificate. This means that if you want to generate a client certificate after, for example, 50 days from the beginning of the CA certificate validity, the client certificate validity period must be with at least 50 days less than the CA certificate validity period.

It is convenient to create a copy of the OpenSSL configuration file (e.g. `openssl_cl.cnf`) and modify its parameters to be specific for client certificates generation.

5.4.1 Creating the client Certificate Signing Request

The command is the same as the one presented in section **5.3.1 Create the server Certificate Signing Request**:

```
openssl req -new -config ./openssl_cl.cnf -newkey rsa:1024 -md5 -nodes  
-keyout ./cr/CL01key.pem -out ./cr/CL01req.pem -days 300
```

The only differences are the configuration file used (`./openssl_cl.cnf`), which is configured to be used for client certificates generation and the names of the files for storing the private key (`./cr/CL01key.pem`) and the Certificate Signing Request -out (`./cr/CL01req.pem`).

A sample output is presented below. The lines that require you to enter some information are outlined in boldface:

```
Using configuration from ./openssl_cl.cnf  
Generating a 1024 bit RSA private key  
.+++++  
.....+++++  
writing new private key to './cr/CL01key.pem'  
-----  
You are about to be asked to enter information that will be  
incorporated  
into your certificate request.  
What you are about to enter is what is called a Distinguished Name or a  
DN.  
There are quite a few fields but you can leave some blank  
For some fields there will be a default value,  
If you enter '.', the field will be left blank.  
-----  
Country Name (2 letter code) [FI]:  
State or Province Name (full name) [Helsinki]:  
Locality Name (eg, city) [Helsinki]:  
Organization Name (eg, company) [MyCompany]:  
Organizational Unit Name (eg, section) [MyCompany HQ]:  
Common Name (eg, your name or your server's hostname) []:JohnDoe  
Email Address [root@company.com]:  
  
Please enter the following 'extra' attributes  
to be sent with your certificate request  
A challenge password []:  
An optional company name []:
```

IMPORTANT: keep the generated files in secure locations, otherwise anyone could generate fake certificates in your name. The files should be readable only by the system administrator. For example, on UNIX systems, the files should be owned by root and the permission mode should be set to 400.

5.4.2 Self-signing the client Certificate Signing Request

The command is similar with the one presented in section **5.3.2 Self-signing the server Certificate Signing Request**:

```
openssl ca -config ./openssl_cl.cnf -policy policy_match -days 300  
-md md5 -out ./cr/CL01cert.pem -keyfile ./cr/CAkey.pem -outdir ./cr  
-cert ./cr/CAcert.pem -infiles ./cr/CL01req.pem
```

A sample output is presented below. The lines that require you to enter some information are outlined in boldface:

```
Using configuration from ./openssl_cl.cnf  
Enter PEM pass phrase:  
Check that the request matches the signature  
Signature ok  
The Subjects Distinguished Name is as follows  
countryName          :PRINTABLE:'FI'  
stateOrProvinceName  :PRINTABLE:'Helsinki'  
localityName         :PRINTABLE:'Helsinki'  
organizationName     :PRINTABLE:'MyCompany'  
organizationalUnitName:PRINTABLE:'MyCompany HQ'  
commonName           :PRINTABLE:'JohnDoe'  
emailAddress         :IA5STRING:'root@company.com'  
Certificate is to be certified until Mar 28 11:30:17 2002 GMT (300  
days)  
Sign the certificate? [y/n]:y  
  
1 out of 1 certificate requests certified, commit? [y/n]y  
Write out database with 1 new entries  
Data Base Updated
```

The PEM pass phrase is the one that you defined when creating your CA certificate (see section **5.2.1 Being your own CA**). Don't forget that the certificate validity period should be completely inside the validity period of the CA certificate.

The extra operations presented in the section **5.3.2 Self-signing the server Certificate Signing Request** after signing the certificate are no longer required here, except from copying the `xx.pem` file into the `demoCA/newcerts` directory. From this point you should continue with section **5.4.4 Exporting the client certificate**.

IMPORTANT: keep the generated files in secure locations, otherwise anyone could generate fake certificates in your name. The files should be readable only by the system administrator. For example, on UNIX systems, the files should be owned by root and the permission mode should be set to 400.

5.4.3 Signing the client Certificate Signing Request by 3rd party CAs

The procedure for signing the client Certificate Signing Request is the same as the one presented in section 5.3.3 **Signing the server Certificate Signing Request by 3rd party CAs**, except that you don't have to copy the content of the private key and certificate files into a new file.

5.4.4 Exporting the client certificate

As it was said earlier, the client applications don't usually use the PEM format for client certificates, therefore you have to export them in a supported format, such as PKCS#12 (also known as PFX) or DER. In most cases you will use PKCS#12 format, but certain applications require DER format certificates. One such example is the Outlook Express program, which can't import PKCS#12 certificates. However, if you import PKCS#12 certificates in Internet Explorer, they can be used from Outlook Express as well.

5.4.4.1 Exporting client certificates in PKCS#12 (PFX) format

To export the client certificate in PKCS#12 format, you can use a command like the one below:

```
openssl pkcs12 -export -in ./cr/CL01cert.pem -inkey ./cr/CL01key.pem  
-des3 -descert -certfile ./cr/CAcert.pem -name "JohnDoe" -out  
./cr/CL01.p12
```

The parameters have the following meaning:

| | |
|--------------------------------------|---|
| <code>pkcs12</code> | The OpenSSL utility for managing PKCS#12 certificates. |
| <code>-export</code> | Specifies that a certificate must be exported. |
| <code>-in ./cr/CL01cert.pem</code> | Specifies the client certificate file. |
| <code>-inkey ./cr/CL01key.pem</code> | Specifies the file containing the private key for the client certificate. |
| <code>-des3</code> | Use Triple DES to encrypt the private key before putting it in the exported certificate. Other options are <code>-des</code> (DES cipher), <code>-idea</code> (IDEA cipher) or <code>-nodes</code> (no encryption). |
| <code>-descert</code> | Use Triple DES to encrypt the certificate. If this option is not used, the default cipher for certificates will be used (40 bit RC2). |

| | |
|--|---|
| <code>-certfile ./cr/CAcert.pem</code> | The CA certificate file used to authenticate the client certificate |
| <code>-name "JohnDoe"</code> | The name of this certificate. |
| <code>-out ./cr/CL01.p12</code> | The file in which the exported certificate will be placed. |

If you want, you can be more specific in setting the ciphers used for encrypting the private key and the certificate. Instead of using the `-des3` and `-descert`, you can use the `-keypbe` and `-certpbe` options, each followed by the name of the ciphers to be used for the private key and certificate respectively. The possible cipher names are:

- **PBE-MD2-DES, PBE-MD5-DES**
They only offer 56 bits of protection since they both use DES.
- **PBE-SHA1-RC2-64, PBE-MD2-RC2-64, PBE-MD5-RC2-64, PBE-SHA1-DES**
They use either 64 bit RC2 or 56 bit DES encryption.
- **PBE-SHA1-RC4-128, PBE-SHA1-RC4-40, PBE-SHA1-3DES, PBE-SHA1-2DES, PBE-SHA1-RC2-128, PBE-SHA1-RC2-40**
These algorithms use the PKCS#12 password based encryption algorithm and allow strong encryption algorithms like triple DES 168 bit or 128 bit RC2/RC4 to be used.

You will be asked to provide an export password. You will have to provide this password to the users together with certificates, otherwise they will not be able to import this certificates in their client applications. As the password is used for encrypting the certificate, it's advisable to choose long passwords.

The export password is a very good security mechanism. Even if someone steal one of your client certificates, the certificate cannot be used without the password. But this also implies that you have to educate the users to be careful with their passwords and to not divulge them to others or write them down.

Also, the administrator must be careful how she/he distributes the certificates and passwords to the users. It is not recommended to distribute them by e-mail, as e-mails can be intercepted. In the worst case, you could send only the certificate file by e-mail, and then find a way to personally communicate the password to the user.

5.4.4.2 Exporting client certificates in DER format

In order to export the client certificate in DER format, you first have to copy the contents of the files `cr/CL01key.pem` and `cr/CL01cert.pem` (created in sections **5.4.1 Creating the client Certificate Signing Request** and **5.4.2 Self-signing the client Certificate Signing Request**) into a single file, for example `cr/CL01.pem`.

After that, run the following command (all in one line):

```
openssl x509 -in ./cr/CL01.pem -out ./cr/CL01.cer -outform DER
```

Please note that the DER format certificate file `CL01.cer` is not password protected, therefore is much less secure than a PKCS#12 certificate.

5.4.5 Converting client certificates from PKCS#12 to PEM

The client certificates can be converted from PKCS#12 format back into PEM format.

This function could be used, for example, for fetching a certificate from a smartcard and converting it to the PEM format, which can then be inserted in the list of trusted certificates. You could get the client certificate either by using the smartcard reader software (if it allows it) or by exporting it from an application (e.g. a browser) that has imported the certificate from that smartcard.

If the certificate is password protected, you will need the password in order to export it. You don't need to export the private key too, only the certificate is important since this is where the public key and identification data are stored. Exporting the private key is a security risk, and users should be instructed to avoid it.

To convert a PCKS#12 file to PEM format, you can use a command like:

```
openssl pkcs12 -in ./CL02.p12 -out ./CL02.pem -clcerts
```

This command gets the client certificate or certificates from the file `./CL02.p12` and exports them to the file `./CL02.pem`. You will be prompted for the export password of the `./CL02.p12` certificate. If this file also contains a private key, you will be prompted for a new PEM pass phrase to be used for encrypting the exported private key, which will be placed in the `./CL02.pem` file as well.

5.5 Creating private keys.

If you need them, it is possible to create private keys outside the process of certificate generation.

In most cases, you will want to create RSA private keys. To do that, run in the OpenSSL installation directory the following command:

```
openssl genrsa -des3 -out ./cr/rsakey.pem 1024
```

The parameters have the following meaning:

| | |
|----------------------|---|
| genrsa | The OpenSSL utility used for generating RSA keys |
| -des3 | The cipher used for encrypting the private key (Triple DES). Instead of -des3, you can use the -des or -idea switches, for selecting DES or IDEA ciphers. If you don't use this option, no encryption will be used. However, if you use it, you will be asked for PEM pass phrase, which you will need to enter every time you use the private key. |
| -out ./cr/rsakey.pem | The file used for storing the private key. |
| 1024 | The size of the generated private key, in bits. It must be the last option specified in the command. The default value is 512, but 1024 is more recommended. |

You will see an output like the one below. The lines marked with boldface will ask for your input.

```
Generating RSA private key, 1024 bit long modulus
.....+++++
.....+++++
e is 65537 (0x10001)
Enter PEM pass phrase:
Verifying password - Enter PEM pass phrase:
```

The PEM pass phrase you must remember, as you will need it every time you will use this private key, e.g. for generating certificates.

A RSA key is suitable for most uses. However, in case you need to use a DSA key instead of RSA, you can use a command similar with the one for generating the RSA key:

```
openssl gendsa -des3 -out ./cr/dsakey.pem ./cr/dsakeyparams.pem
```

The differences are that the `gensa` utility is used instead of `genrsa` and instead of specifying the number of bits, you need to specify a DSA parameters file. This file can be generated with a command like:

```
openssl dsaparam -out ./cr/dsakeyparam.pem 1024
```

These private key keys could also be used in the process of generating certificates. To do that, instead of using the `-newkey` and `-keyout` parameters when creating Certificate Signing Requests or self-signed certificates (which are used to generate new private keys), you can use the `-key rsakey.pem` parameter, which means that an existent private key (stored in the `rsakey.pem` file) should be used.

IMPORTANT: keep the PEM pass phrase and the generated files in secure locations, otherwise anyone could generate fake certificates in your name. The files should be readable only by the system administrator. For example, on UNIX systems, the files should be owned by root and the permission mode should be set to 400.

6 Using Stunnel

6.1 Integrating Stunnel with Teamware Office services

The most common usage for Stunnel is to set it as an SSL wrapper around a non-SSL port. A common usage for Stunnel looks like this:

```
stunnel -d 443 -r 80 -p /certs/SRV.pem -D 4
```

The parameters used have the following meaning:

- `-d 443` The TCP/IP port number (443 in this example) that Stunnel will open and on which it will listen for incoming SSL requests.
- `-r 80` The TCP/IP port number (80 in this example) to which Stunnel will forward the decrypted requests and from which it will get the responses. That port could be, for example, the port number used by Teamware Office Web Service.
- `-p /certs/SRV.pem` The server certificate that Stunnel will present to SSL clients. For information about generating server certificates, please see section **5.3 The server certificate**. Of course, you will have to provide the correct path for the certificate file.
- `-D 4` Sets the level of logging that Stunnel will produce. The possible options are: 0 - emergency messages, 1 - alerts, 2 - critical messages, 3 - error messages, 4 - warnings, 5 - trace messages, 6 - info or 7 - debug. The default level is 5, which is the level you will want to use in the beginning, to make sure that everything works as expected. After that, you will probably want to use the level 4, so only messages of "warning" importance or higher will be logged.

Stunnel does not need to know what application is listening on the port specified by `-r`, or what communication protocol (HTTP, IMAP, POP3 etc.) is the application using for that port. All it does is to decrypt the SSL requests coming on the port specified by `-d`, decrypt them and pass them to the port number specified by `-r`. After that, Stunnel will get the responses from the remote port, encrypt them and pass them to the SSL client.

Of course, behind the scene the things are a little more complicated than that. See the section **2 About SSL** for details.

The sample command that was presented above can be used with any of the following Teamware Office services:

- Web Service (HTTP)
- IMAP
- POP3
- SMTP
- NNTP
- LDAP

The table below shows the default port numbers for these services (to be used with option `-r`) and their default SSL counterparts (to be used with `-d` option):

| Protocol | Default port number (Teamware Office) | Default SSL port number (Stunnel) |
|-------------|---------------------------------------|-----------------------------------|
| HTTP | 80 | 443 |
| IMAP | 143 | 993 |
| POP3 | 110 | 995 |
| SMTP | 25 | 465 |
| NNTP | 119 | 563 |
| LDAP | 389 | 636 |

Of course, in your systems the Teamware Office services might be using other port numbers than the default ones, so specify your own port numbers if they differ from the one presented in this document. The SSL port numbers presented are the ones established as being the standard default ones. You are free to use different port numbers, as long as your user know which they are so they can set their client application accordingly.

Additionally, especially if your server has multiple IP addresses, you can specify the address to be used for binding. Here are two examples:

```
stunnel -d 443 -r 127.0.0.1:80 -p /certs/SRV.pem  
stunnel -d 172.30.1.3:993 -r 172.30.1.1:143 -p /certs/SRV.pem
```

In the first example, Stunnel will open the port 443 (HTTPS) on all available interfaces, but will use only the interface corresponding to the IP address 127.0.0.1 (lo) for connecting to the server that listens on the port 80 (Web Service).

In the second example, Stunnel will open the port 993 (SSL IMAP) only on the interface corresponding to the IP address 172.30.1.1, and it will use the same address for connecting to remote Teamware Office IMAP service.

Instead of IP addresses, you can use machine names as well, for example:

```
stunnel -d two.company.com:993 -r two.company.com:143 -p /certs/SRV.pem
```

For the IMAP, POP3, NNTP and LDAP services there are no Teamware Office specific configurations required (except, of course, starting these services) if you want to provide SSL encryption for them using Stunnel.

Only the Web Service requires some additional configuration operations, presented in section **6.7 Integrating Stunnel with the Teamware Office Web Service**). The SMTP service doesn't require extra configuration, but please read the notes from section **6.8 Using Stunnel with the Teamware Office MIME Connector**

The usage mode presented so far for integrating Stunnel with the Teamware Office it is not the only one possible. The following paragraphs will present additional configuration options.

6.2 Providing both SSL and non-SSL services

It is important to note that Stunnel will not block the ports of the Teamware Office services for which it provides SSL encryption. In other words both the non-SSL (Teamware Office) and SSL (Stunnel) ports will be accessible.

You could, for example, allow connections coming from the company's LAN or VPN to go directly to the Teamware Office ports, but all connections from the Internet will have to go through Stunnel's ports.

This can be achieved by configuring the firewall to allow or reject connection to the Teamware Office or Stunnel ports depending on the origin of the connection.

These kinds of configurations are useful when you want to reduce the server load by not requiring SSL encryption/decryption for connections from secure networks.

Of course, if your security requirements are so, you can block access to the Teamware Office services for all clients. You will only need to allow the connections coming from the machines running Stunnel.

Of course, if you don't have a firewall, you could look for other possibilities for disabling connections from insecure networks to the non-SSL ports. But if you don't have a firewall your network is already vulnerable and just by adding SSL encryption to the Teamware Office services it will not make it much more secure.

6.3 Automatically starting Stunnel

On UNIX and Linux systems, the most simple solution is to add the Stunnel startup command (or commands, if you want to encrypt multiple ports) in your one of your local

startup scripts, for example, `rc.local`. The Stunnel events will be logged to the standard `syslogd` daemon.

On Windows systems, unfortunately, Stunnel is not yet able to run as a service, at least not in the binary distribution form. Craig Boston wrote a patch (the source code can be downloaded from <http://www.stunnel.org/patches>) which enables Stunnel to run as a Windows service. If you have the Windows Resource Kit, you can also look at using the `srvany.exe` application, which should allow you to run Stunnel as a service.

As a more simple solution, you could use the Task Scheduler, available from Control Panel\Scheduled Tasks.

To use this solution, create a batch command file (for example `stunnel.cmd`) which should contain the Stunnel startup commands, for example (everything in one line):

```
c:\stunnel\stunnel.exe -d 443 -r 80 -p d:\openss\demoCA\SRVstunnel.pem  
2> c:\stunnel\stunnel_http.log
```

This command starts Stunnel, as already presented. Additionally, it will create an Stunnel log file (`stunnel_http.log`). Please note that `2>` is used for redirecting the output to the log file, because Stunnel will output its messages to `StdErr`. To keep the log files between system restarts, use `2>>` instead of `2>`. If you want to use Stunnel for multiple services, add similar lines for each service.

After you have created the batch file, use the Task Scheduler to schedule this batch file to be run at system startup, under a system administrator account. After creating the new task, open the task properties window and disable the "Stop the task if it runs for..." option in the Setting tab.

6.4 Running Stunnel on other machines than Teamware Office

It is not necessary to run Stunnel on the same machine or machines on which Teamware Office is running. Stunnel can open an SSL port on a machine and connect to a remote port on another machine. These machines don't need to run the same operating system.

For example, you could run Stunnel on the firewall server (e.g. `firewall.company.com`) and connect it to remote services on a Teamware Office server, for example:

```
stunnel -d 443 -r two1.company.com:80 -p /certs/SRV.pem  
stunnel -d 444 -r two2.company.com:80 -p /certs/SRV.pem
```

These commands will open two SSL ports (443 and 444) on the firewall machine, and all requests coming to these ports will be forwarded (after decryption) to the Teamware Office Web Services on the machines `two1.company.com` and `two2.company.com` respectively.

Please note that, using such a configuration, you can provide access from Internet to the Teamware Office services, without needing public IP addresses for the Teamware Office servers. Only the firewall machine needs a public IP address. This greatly enhances the security of the Teamware Office servers, as they are hidden in the internal network.

6.5 Specifying the ciphers to be used by Stunnel

When an SSL client connects to Stunnel, they will negotiate to use the most secure ciphers that both of them support. In most cases this is not a problem, if recent SSL clients (mail clients, Web browsers etc.)

However, if you have very sensitive information on your servers and don't want to induce security risks because of users using old software, you can force Stunnel to allow the usage of only of specific ciphers.

Of course, you can use the same method if you are more interested in speed of communication than on data security: you can force Stunnel to use only weak ciphers, which are a little bit faster.

To get the list ciphers that are provided by OpenSSL on your system, you can use the following command:

```
openssl ciphers -v
```

On a system running OpenSSL 0.9.6, you should see a list like:

```
EDH-RSA-DES-CBC3-SHA      SSLv3 Kx=DH      Au=RSA  Enc=3DES(168) Mac=SHA1
EDH-DSS-DES-CBC3-SHA     SSLv3 Kx=DH      Au=DSS  Enc=3DES(168) Mac=SHA1
DES-CBC3-SHA             SSLv3 Kx=RSA     Au=RSA  Enc=3DES(168) Mac=SHA1
DES-CBC3-MD5             SSLv2 Kx=RSA     Au=RSA  Enc=3DES(168) Mac=MD5
DHE-DSS-RC4-SHA         SSLv3 Kx=DH      Au=DSS  Enc=RC4(128)  Mac=SHA1
RC4-SHA                  SSLv3 Kx=RSA     Au=RSA  Enc=RC4(128)  Mac=SHA1
RC4-MD5                  SSLv3 Kx=RSA     Au=RSA  Enc=RC4(128)  Mac=MD5
RC2-CBC-MD5              SSLv2 Kx=RSA     Au=RSA  Enc=RC2(128)  Mac=MD5
RC4-MD5                  SSLv2 Kx=RSA     Au=RSA  Enc=RC4(128)  Mac=MD5
RC4-64-MD5               SSLv2 Kx=RSA     Au=RSA  Enc=RC4(64)   Mac=MD5
EXP1024-DHE-DSS-RC4-SHA SSLv3 Kx=DH(1024) Au=DSS  Enc=RC4(56)   Mac=SHA1 export
EXP1024-RC4-SHA          SSLv3 Kx=RSA(1024) Au=RSA  Enc=RC4(56)   Mac=SHA1
export
EXP1024-DHE-DSS-DES-CBC-SHA SSLv3 Kx=DH(1024) Au=DSS  Enc=DES(56)   Mac=SHA1
export
EXP1024-DES-CBC-SHA      SSLv3 Kx=RSA(1024) Au=RSA  Enc=DES(56)   Mac=SHA1
export
EXP1024-RC2-CBC-MD5     SSLv3 Kx=RSA(1024) Au=RSA  Enc=RC2(56)   Mac=MD5
export
EXP1024-RC4-MD5         SSLv3 Kx=RSA(1024) Au=RSA  Enc=RC4(56)   Mac=MD5
export
EDH-RSA-DES-CBC-SHA     SSLv3 Kx=DH      Au=RSA  Enc=DES(56)   Mac=SHA1
EDH-DSS-DES-CBC-SHA     SSLv3 Kx=DH      Au=DSS  Enc=DES(56)   Mac=SHA1
DES-CBC-SHA             SSLv3 Kx=RSA     Au=RSA  Enc=DES(56)   Mac=SHA1
DES-CBC-MD5             SSLv2 Kx=RSA     Au=RSA  Enc=DES(56)   Mac=MD5
EXP-EDH-RSA-DES-CBC-SHA SSLv3 Kx=DH(512)  Au=RSA  Enc=DES(40)   Mac=SHA1 export
```

| | | | | | | |
|-------------------------|-------|-------------|--------|-------------|----------|--------|
| EXP-EDH-DSS-DES-CBC-SHA | SSLv3 | Kx=DH(512) | Au=DSS | Enc=DES(40) | Mac=SHA1 | export |
| EXP-DES-CBC-SHA | SSLv3 | Kx=RSA(512) | Au=RSA | Enc=DES(40) | Mac=SHA1 | export |
| EXP-RC2-CBC-MD5 | SSLv3 | Kx=RSA(512) | Au=RSA | Enc=RC2(40) | Mac=MD5 | export |
| EXP-RC4-MD5 | SSLv3 | Kx=RSA(512) | Au=RSA | Enc=RC4(40) | Mac=MD5 | export |
| EXP-RC2-CBC-MD5 | SSLv2 | Kx=RSA(512) | Au=RSA | Enc=RC2(40) | Mac=MD5 | export |
| EXP-RC4-MD5 | SSLv2 | Kx=RSA(512) | Au=RSA | Enc=RC4(40) | Mac=MD5 | export |

The first column shows the name of the cipher in the form that it has to be passed to Stunnel. The second column shows the SSL version (2 or 3) for which the cipher is available; SSL v3 also include the TLS ciphers. The next columns show the ciphers used for the key exchange, authentication, certificate encryption and digest.

For specifying the ciphers to be used by Stunnel, you will have to use the `-c` option, followed by the list of allowed ciphers, separated by colons, for example:

```
stunnel -d 443 -r 80 -p /certs/SRV.pem -c DES-CBC3-SHA:DES-CBC3-MD5
```

The above command will set Stunnel to use only Triple DES encryption (168 bit), with MD5 or SHA digests and RSA (up to 1024 bit) key exchange. Of course, this is only an example. In real life, you will want allow more ciphers, depending on the your security requirements and the type of clients you want to allow to connect.

6.6 Restricting access using client security certificates

Section 5.4 **Client certificates** presented the procedure for creating client certificates. In this section you will see how to use these certificates.

6.6.1 Server-side configurations for client certificates

In order to be able allow/restrict access using client certificates, you will have to specify to the Stunnel server which client certificates to accept.

Here is an example:

```
stunnel -d 443 -r 80 -p /certs/SRV.pem -v 2 -a /trusted -S 0 -t 300
```

```
stunnel -p /usr/share/ssl/certs/SRV.pem -v 3 -a /usr/share/ssl/trusted -d 446 -r 1080 -S 0 -t 60
```

The `-d`, `-r` and `-p` options have already been presented. The new options are:

- `-v 2` Specifies the level of verification to be used. The possible values are:
- 1 - verify peer certificates, if present
 - 2 - verify peer certificates every time
 - 3 - verify peer certificates every time against a local list of trusted certificates

`-a /trusted` Specifies the directory in which the trusted CA certificates and (if `-v 3` is used) the trusted client certificates are stored. Instead of `-a`, you can also use the `-A` option, in which case you will specify a specific trusted certificate file instead of a directory.

`-s 0` Sets the defaults for the certificate sources. Possible values are:

- 0 - Ignore all defaults
- 1 - Use OpenSSL defaults
- 2 - Use Stunnel defaults
- 3 - Use Stunnel and OpenSSL defaults

It is better to use `-s 0` and the `-a` or `-A` options, to be sure that only the certificates you want will be used.

`-t 300` Sets the session cache time-out. When a client connects to Stunnel first time, its session information will be stored in the session cache for the number of seconds specified by `-t`. If client information exists in cache, subsequent connections from that client will not go through the entire SSL handshake process again.

If `-v 2` is used, Stunnel will ask the client SSL applications to present a valid client certificate signed by a CA whose certificate is stored in the location specified by the `-a` or `-A` option. If the client doesn't present such a certificate, the connection will be refused.

The option `-v 3` is useful if you want to allow access only to clients holding certain client certificates. If `-v 3` is used, the same verifications as above will be performed. But, additionally, the directory or file specified by `-a` or `-A` must contain a copy of the client certificate presented by the client, otherwise the connection is rejected.

With option `-v 1`, same verifications will be performed as with `-v 2`, but only if the client presents a certificate. If the client doesn't have a client certificate, it will be allowed to connect.

If you will be using option `-a`, copy the CA certificate (for example the `CAcert.pem` file from the section **5.2.1 Being your own CA**) in the directory designated by `-a`. Additionally, if you intend to use the `-v 3` option, you will have to copy in that directory the PEM form (not PKCS#12 or DER) of the certificates (i.e. the file `CL01cert.pem` from section **5.4 Client certificates**).

After you have placed the certificates in the trusted directory, go to that directory and issue the command:

```
c_rehash ./
```

This will generate the symbolic links needed by Stunnel and OpenSSL in order to identify the certificates in that directory. Please note the `./` used, to designate the current

directory. If you forget to specify it, the default `certs` directory of OpenSSL will be rehashed.

If you want to use the `-A` option, you will have to create a single file that will store all the CA certificates and (if using `-v 3`) all the client certificates that you choose to start. The CA certificates should be stored at the top of the file.

It is advisable that, after adding or deleting client certificates from the trusted file or directory, to restart Stunnel.

6.6.2 Client-side configurations for client certificates

In order for the users to be able to use the client certificates that have been issued for them, they must install them in their client applications. The following sections will present the procedure for some of the most used SSL clients. If your client application is not presented here, please consult the documentation of that application

6.6.2.1 Using client certificates with Internet Explorer

Open the Internet Options window from the Tools menu, and then in the Content tab click on the Certificates... button.

The Certificates window will open, showing the certificates that you hold (Personal tab), the certificates that you have from other persons (Other People tab), the CAs that you have chosen to trust only for a session (Intermediate Certificate Authorities tab) and the CAs that you have chosen to always trust (Trusted Root Certification Authorities tab).

To import a certificate, click on the Import button on any of these tabs. You will be first prompted for the file certificate that you want to import (see section **5.4.4 Exporting the client certificate** for details on generating these certificates).

After selecting the file containing the certificate, you will be taken a new window, where you will have to enter the password for that certificate. It is also advisable to enable the "Enable strong private key protection" check box. This way, you will be prompted for approval any time an application wants to use this certificate. It prevents the risk of malicious applications using your identity without your knowledge.

Then you will be asked for a certificate store. You should let Internet Explorer to automatically choose the proper stores fore the certificate. After that, answer Yes/OK to the rest of the question.

You should now be able to see the imported certificate in the Personal tab, and the CA that signed this certificate in the Trusted Root Certification Authorities tab. You can use the Advanced button for viewing more details about the certificates.

After you have successfully imported the certificate, you should be able to access the Teamware Office services that are SSL protected by Stunnel using client certificate validation.

Please note that importing a client certificate does not remove the certificate file from your disk. So, even if the certificate file is password protected, make sure that you store it in a safe location. Better yet, you should completely delete it from your disk. If in the future you will need it again, you can ask for a new copy from the system administrator.

If you have enabled the "Enable strong private key protection" check box, you will be prompted for confirmation every time you will need to use the certificate to identify yourself. If you have multiple client certificates installed, you will be also asked to choose which one to use for identifying to the site that are you trying to open.

Please note that the certificates that you import in Internet Explorer can be used from other Microsoft applications as well, such as Outlook, for all the services they can access (IMAP, POP3, NNTP etc.).

If you want to remove the imported certificates, simply open the Certificates window again, select the certificate that you want to delete and press the Remove button. You can choose to remove only the client certificate (Personal tab), only the CA information (Trusted Root Certificate Authorities tab) or both. After removing them, you should restart Internet Explorer and any other application that uses these certificates.

6.6.2.2 Using client certificates with Netscape

Click on the Security button from the main Netscape toolbar. On the left side of the new window that opens, click on Yours under Certificates, which will open the section showing your certificates.

Click on the Import a Certificate button and choose the file storing the certificate that you want to import. After entering the certificate password, the certificate will be imported.

But before you can use it, you must enable it. Click on Signers under Certificate on the left side of the window. Select from the list of CAs the one that has signed the certificate, and then click on the Edit button. In the new window that opens, select the types of activities for which you want to allow the certificate to be used for. Normally, you will want to enable everything: Certifying network sites, Certifying e-mail addresses and Certifying software developers. Of course, the available options will depend on the settings that were used when generating the certificate.

Please note that, if you have imported more certificates signed by the same CA, when pressing the Edit button in the Signers section you will be first be prompted to select the client certificate that you want to edit. Unfortunately, the list will not show the actual name of the certificates, it will only show the validity dates for the available certificate.

Netscape will prompt you every time you want to access a site that requires a client certificate, asking if you still want to connect and which certificate to use if more are available.

To delete a client certificate from Netscape, click again on the Security button from the main toolbar. Then select the certificate that you want to delete from the Yours or Signers sections and press Delete.

6.6.2.3 Using client certificates with Mozilla

Click on the Security Information button on the bottom right corner of the Mozilla window (the button has only the picture of a lock on it).

In the Personal Security Manager window go to the Certificates tab, Mine section and press on the Restore button to import a client certificate. After choosing the certificate file and entering the certificate password, the certificate will show up on the list with your client certificates.

Go then to the Authorities section, select from the list the CA that signed your client certificate and press on the Edit button to select what types of activities do you want to allow for the certificates signed by this CA.

Mozilla, by default, will prompt for your confirmation and for a certificate to use every time you try to access a site that requires client certification. You can disable this feature in the Personal Security Manager window, Applications tab, Navigator section, although it is not advisable to do it.

Please also note that certain versions of Mozilla have a rather strange way of handling SSL sessions. More specifically, Mozilla will prompt for your confirmation of the client certificate usage for each element (e.g. frames, images etc.) that it fetches from an SSL encrypted page.

To delete certificates from Mozilla, open the Personal Security Manager window, select the certificates that you want to delete from the Mine or Authorities sections and press the Delete button.

6.7 Integrating Stunnel with the Teamware Office Web Service

As it was said previously in this document, if you want to provide SSL encryption for the Teamware Office Web Service, certain extra configuration steps are needed apart from setting up Stunnel and OpenSSL.

In certain templates, Teamware Office Web Service uses macros that are expanded at run-time to the URL of a certain object. Since these objects can be located on other Teamware Office servers, those URLs will contain the default protocol and port number for the Web Service on that server. This means that those URL will use the Teamware Office HTTP protocol and port, not the HTTPS protocol and the Stunnel port number.

The solution is very simple: to use the [Redirect] section of http.ini. Such a section would look like:

```
[Redirect]
  direct = 127.0.0.1
  https://two.company.com/* = all
```

The first line specifies that requests coming from the IP address 127.0.0.1 (localhost) should be accepted directly, without any redirection. The second line specifies that requests coming from any other IP address should be redirected to the address https://two.company.com, where two.company.com is the machine on which Stunnel is running.

Important note:

If the Stunnel port number connected to the Teamware Office Web service is 443, do NOT specify the port number in the [Redirect] section, because 443 is the default port number for the HTTPS protocol. Specify the port number only if it anything else than 443, for example:

```
https://two.company.com:600/* = all
```

If Stunnel is running on the same machine as the Teamware Office Web Service (which uses port 80 in this example), it should be started with a command like:

```
stunnel -d 443 -r 127.0.0.1:80 -p /certs/SRV.pem
```

The line `direct=127.0.0.1` from the [Redirect] section has two effects:

- It prevents Stunnel and Web Service from creating endless loops (requests being passed from Stunnel to the Web service and then redirected back to Stunnel and so on)
- Teamware Office Web Service sometimes uses HTTP requests to fetch data from itself. These requests must be allowed to go directly to the Web Service HTTP port instead of being redirected to the Stunnel HTTPS port.

If Stunnel and the Teamware Office Web Service are running on different machines, Stunnel should be started with a command like:

```
stunnel -d 443 -r 172.30.1.2:80 -p /certs/SRV.pem
```

The redirect section should look like:

```
[Redirect]
  direct = 127.0.0.1
  direct = 172.30.1.1
  https://stunnel.company.com/* = all
```

In this example, Teamware Office Web Service is running on the machine 172.30.1.2 and Stunnel is running on the machine stunnel.company.com with the IP address 172.30.1.1.

The `direct` option can be also used to allow requests coming from certain networks to go directly to the Teamware Office HTTP port instead of being redirected through Stunnel. For example, if you want to allow HTTP connections from the network 172.30.1.*, just add in the `[Redirect]` section a line like:

```
  direct = 172.30.1.
```

Important note:

The standard templates `mail/list0.*` that come with some releases of Teamware Office 5.4 Ed. 1 contain a line that starts with:

```
<!--#tw get_url="$protocol;://$hostname;/mail.....
```

For the Teamware Office Web Services which will be using Stunnel, you must delete the part `$protocol;://$hostname;`, so the line will start with:

```
<!--#tw get_url="/mail.....
```

6.8 Using Stunnel with the Teamware Office MIME Connector

The SMTP protocol is used in two main applications:

- Used in communications between a mail client and a mail server, when the mail client wants to send a message.
- Used in communications between servers for delivering mail messages.

Currently SSL SMTP is used mostly for client-server communications and this is the subject that will be discussed here. SSL communications between mail servers are not so widely used at this moment, because of the big diversity of SMTP servers on Internet.

However, for the close future, you could consider implementing inter-server SSL SMTP communications inside your company or with certain partners.

If you have configured Teamware Office Connector for MIME so your users can use it as an outgoing SMTP server for their mail clients, you can use Stunnel to provide SSL encryption for the SMTP port used by Connector for MIME, with the commands that were presented in previous sections.

Please note that, by default, most SMTP clients can't use client certificates for identifying themselves to the SMTP servers, so don't use the `-v` option when starting Stunnel.

Of course, you can use the same technique for any SMTP server that is used by your clients.

7 Additional resources

The list below presents a number of Internet resources that you might find useful. We take this opportunity to express our gratitude to the people behind them.

- www.stunnel.org - The Stunnel Web site
- www.openssl.org - The OpenSSL Web site
- <http://developer.netscape.com> - Contains a lot of documentation about the SSL protocol.
- <http://www.ietf.org/rfc/rfc2246.txt?number=2246> - IETF draft specifications for the new TLS protocol.
- <http://www.drh-consultancy.demon.co.uk> - The home page of Dr. Stephen N. Henson, the author of the PKCS#12 package, currently part of the OpenSSL standard distribution.
- www.openca.org - The OpenCA project, a very good SSL resource.
- www.verisign.com - Very known CA and provider of cryptography solutions. Trial certificate signatures available.
- www.rsa.com - Very known CA and provider of cryptography solutions. The developer of the RSA algorithm.
- www.thawte.com - Very known CA and provider of cryptography solutions. Trial certificate signatures available.