Zeus Technology

SSL: Theory and Practice

Date:	16th June 2000
Version:	1.0

Zeus Technology
Newton House
Cambridge Business Park
Cowley Road
Cambridge
CB4 0WZ
England
Phone: +44 (0) 1223 525000
Fax: +44 (0) 1223 525100
Email: info@zeus.com
Web: http://www.zeus.com

Table of Contents

0 About this document

This document introduces the theory and practice of creating an SSL secured website.

It begins with an in depth discussion of the need for security, and describes the theory that underlies SSL authentication and encryption. It makes judgements on best practice and the surrounding security context.

It then describes in detail how to configure a Zeus webserver for SSL transactions. This 'configuration' section is largely self contained and can be used as a configuration reference without substantial reference to the previous chapters.

The document is intended for systems integrators and security architects working with Zeus Webserver products and related technologies.

1 Introduction

The Internet is an open system. The identity of people, companies and computers communicating on the Internet is not easy to determine and validate. Furthermore, the very communication path is inherently insecure - all communications are potentially open for an eavesdropper to read and modify as they pass between the communicating endpoints.

Internet communications are often likened to the use of postcards in a traditional mail system. If an attacker is in the right place at the right time, he can:

- Read your postcards, thus snooping on your conversation.
- Modify your postcards, thus subverting your conversation.
- Send postcards to you or your correspondent, thus impersonating either party.

Although such compromises are generally rare and difficult to carry out, the great value and sensitivity of some information transferred over the internet can make the effort to compromise the communication worth the potential gain.

Why SSL?

SSL, the Secure Sockets Layer, aims to secure the communication between network applications, providing privacy, authentication and reliability.

The SSL protocol can be usefully employed on almost any TCP/IP based communication, but is most commonly used to secure HTTP - website traffic.

SSL brings three benefits to the network communication.

- **Privacy**: SSL based communications are private. Encryption is used to secure the communications against eavesdroppers.
- **Authentication**: Each peer (communicant) can be authenticated against a shared infrastructure of trusted authorities.
- **Reliability**: SSL provides additional layers of reliability on top of the underlying TCP/IP protocol, to minimise the danger of the communication layer being compromised.

1.1 Example Applications

SSL is very commonly used in scenarios where the communication needs to be authenticated or secured:

- **e-Commerce**: public access e-Commerce sites are invariably protected by SSL. In this case, SSL serves two purposes:
    1. The e-Commerce site's 'public certificate' serves to identify the site, so the end user can be confident that she is not accessing an impersonating site.
    2. The encryption provided by SSL serves to protect sensitive information (such as credit card details) that are transmitted in the course of the communication. Even if an attacker could capture every packet of data exchanged in the communication, he could not discover the full nature of the communication.

       In addition, SSL's reliability features ensure that if an attacker disrupts the communication layer, in-transit damage can be detected.

- **Corporate Extranets and Intranets**: a company may wish to publish information on its Extranet or Intranet, and restrict access to certain employees or partners ('clients'). In this case:
    1. A client's 'public certificate' serves to identify the client, and this can form the basis of a corporate authorization policy.
    2. The SSL encryption once again protects any sensitive information from eavesdropping or in-transit tampering.

1.2 SSL Protocols and Application Support

SSL version 3 is the most widely deployed version of the SSL protocol. It is supported by all major web server and web browser applications:

Web Servers:

- Zeus v3
- Netscape/iPlanet Enterprise Server v4
- Apache v1.3 (with appropriate SSLeay patches)
- Microsoft IIS v5

Web Browsers:

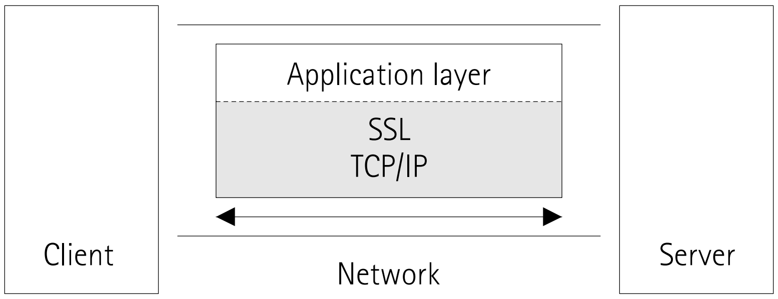
- Microsoft Internet Explorer (all recent versions)
- Netscape Navigator (all recent versions)

An Internet Engineering Task Force (IETF) standard called Transport Layer Security (TLS), based on SSL, has recently been released. This is a refinement of version 3 of the SSL specification. It does not bring any significant additional features or degrees of security over and above those provided by SSL v3, and has not yet been widely adopted.

SSL largely satisfies the requirements and design brief, but has several shortcomings that may be addressed in subsequent versions of the TLS specification.

2 An overview of SSL

SSL is an intermediate network layer, running between the TCP/IP network layer and the higher level application layer (HTTP, IMAP etc.).



Network Layer:

TCP/IP handles delivering network packets from the source to destination. TCP/IP is connection based: one peer (the client) connects to another peer (the server) and establishes a network connection. This connection is used for the duration of the conversation between the two peers. At the end of the conversation, the connection is closed.

Application Layer:

The application layer defines a common, shared protocol that applications can use to communicate over an established TCP/IP connection. For example, HTTP is the protocol that web clients and web servers use to communicate.

An application layer conversation is initiated when a client establishes a TCP/IP connection to a server. Server applications listen on well known 'ports' - for example, HTTP typically runs on port 80 - so a client can commence an HTTP conversation with a server by establishing a TCP/IP connection on the server's port 80.

SSL Layer:

SSL is used to authenticate endpoints and secure the contents of the application level communication. An SSL secured connection begins by establishing the identities of the peers and establishing an encryption method and key in a secure way. The application level communication can then begin. All incoming traffic is decrypted by the intermediate SSL layer and then forwarded on to the application; outgoing traffic is encrypted by the SSL layer before transmission.

In practice, SSL secured servers are typically operated on a different well known port to the overlying application. For example, HTTPS (SSL-secured HTTP) operates on port 443.

3 SSL Authentication

One important goal of SSL is to provide *authentication*. A client connecting to a server will usually want to be able to verify that the server is who they say they are, and not an impostor.

Conversely, a server may wish to establish the identity of the connecting client before allowing the connection to continue. For example, a corporate extranet or a business-to-business application may need to *authorize* each client before allowing them access to sensitive information or ordering systems.

The primary SSL authentication mechanisms depend on a branch of cryptography called 'public key cryptography'.

3.1 Public Key Cryptography

Public Key Cryptography uses an asymmetric pair of keys to encrypt and decrypt data. Each key pair is comprised of a private key and a public key.

Any data that is encrypted with the public key can only be decrypted with the corresponding private key. Conversely, data encrypted with the private key can be decrypted with the public key.

The private key is kept secret, known only to the owner of the key pair. The public key is widely distributed (often as part of a public certificate).

Hence, anyone can encrypt data that only the private key owner can decrypt. It is not possible to decrypt public-key-encrypted data without the private key, except possibly by extreme brute force attacks.

In addition, the owner of the private key can encrypt data that anyone can decrypt. This is commonly used in *Digital Message Signing*. In this scenario, the signer creates a message *digest* (for example, a hash of the message) using an agreed algorithm and then encrypts it with his private key. The recipient can verify that the sender of the message owns the private key by decrypting the message signature with the signer's public key and verifying that the decrypted message digest matches the digest of the message received.

The Public/Private key method of authentication relies on:

- **The security of the private key.** Only the signer of the private key should have access to it. Anyone with access to the private key can impersonate the signer.
- **The validity of the public key.** You need to be absolutely certain that the public key you have received is accurate. If you have the wrong public key, you will be unable to communicate with the signer, and if an impersonator can substitute his public key in the place of the trusted public key, he can impersonate the owner of the private key.
- **The security of the public/private encryption algorithm.** The most commonly deployed public/private key algorithm is RSA's algorithm. Determining the private key given the public key involves complicated mathematics (factoring very large integers), and the strength of a key is related to its size in bits (binary digits).

  RSA regularly run cryptographic challenges, and an RSA key of 512 bits was cracked in August 1999[i]. Computations took 5.2 months using 292 computers and 8,000 MIPS-years of CPU effort. Current wisdom is that, while 512 bit keys are still secure against all but the most extreme cracking attempts, 1024 bit keys are more suitable for securing websites. RSA estimate that 1024 bit keys will remain immune from realistic brute-force cracking for a further 20 years[ii]. Many major Certificate Authorities use 2048 bit keys for their external PKI systems.

Note: The cryptographic strength of Public/Private key pairs is related to their size in bits. 512 bit keys are commonplace, but 1024 bit keys are now being widely deployed. Many web browsers cannot cope with website keys greater than 1024 bits.

3.2 Certificate Authorities

One significant problem with the public/private key system outlined above remains. How can one endpoint of the communication trust that the public key he has been supplied with truly belongs to entity he believes he is communicating with?

The role of a Certificate Authority is significant in establishing this trust. It is an integral part of a Public Key Infrastructure (PKI).

A Certificate Authority (CA) is an intermediate organisation that both parties involved in an SSL communication trust prior to the communication taking place. The role of a certificate authority is verify the identity of an entity who requests a public certificate. This is typically done in an off-line manner, and may be very involved (possibly checking financial and trading records), depending on the policy of the CA.

If the CA can satisfactorily verifies the identity of the requesting entity, it will then issue a digitally signed electronic certificate to the entity. The certificate is signed with the CA's own private key. It is referred to as a *public certificate*.

The entity can then distribute their own public certificate in any means it feels fit. A peer can trust the public key and other details in the certificate if it already trusts the Certificate Authority.

Commonly used public Certificate Authorities include Verisign[iii], Thawte[iv] (now owned by Verisign) and Entrust[v]. Many SSL clients (eg. Web browsers) are preconfigured to trust a number of well regarded CAs such as these.

Many large organisations also operate their own internal Certificate Authority to authenticate employees, networked hardware and other entities. The certificates can be used to authenticate, sign and encrypt not just SSL traffic but email, documents, database accesses and many other digital objects or sessions.

Example

This example uses the `cert` program distributed with Zeus to create and disassemble the PEM-encoded ascii certificate and key files. Other applications can also perform this function - one freely available open source application is `openssl`[vi].

Alice creates a private key and a Certificate Signing Request (CSR). For the purposes of this exercise, the keys are 512 bits long, and the public certificate is to be used to authenticate a machine called `www.alice.zeus.com`.

Alice first generates a private key. Alice keeps this file secret, and does not divulge the contents to anyone.

```
$ cert -new -type private -keysize 512 -out private.key
Generating keys, this may take a few seconds
```

Alice then generates a certificate signing request (CSR). This is a standard format text file which contains the information Alice would like contained in her public certificate. The CSR also contains the public portion of her private key (the public key), and is signed with her private key so the Certificate Authority processing the CSR can verify that the public key portion is valid.

```
$ cert -new -type request -key private.key -out public.csr

The following information is required to make up the certificate.
Optional fields can be left blank by entering a '.'

Country:  GB
State/Province (optional outside US):  .
Locality (town/city):  Cambridge
Organisation:  Zeus Technology
Organisational Unit (optional):  Alice's machine
Common Name (full DNS name of the machine):  www.alice.zeus.com
```

Alice then submits her CSR file to a CA for signing. In this example, we use Thawte's free Test Server Certificate system[vii].

After a short delay, Thawte sends Alice a public certificate generated from the CSR she supplied. In this case, the public certificate is signed by Thawte's test CA root. This test root certificate is not automatically trusted by any client or server software.

In a more realistic scenario, Alice would submit the CSR to her CA along with some accompanying documentation establishing her identity. After further checks, and payment processing for services rendered, the CA would provide Alice with a public certificate signed by a public root certificate that was widely trusted.

We can use the `cert` tool to disassemble Alice's public certificate file, which she has saved in a file named `public.cert`:

```
$ cert -in public.cert -text
X509 Certificate:
    Certificate Info:
        Version: 02
        Serial Number:         12:cd:e0
        Signature Algorithm: md5withRSAEncryption
        Issuer:
            C=ZA, ST=FOR TESTING PURPOSES ONLY, O=Thawte Certification
            OU=TEST TEST TEST, CN=Thawte Test CA Root
        Validity:
            Not Before: Sat, 01 Jan 2000 05:00:00 GMT
            Not After:  Tue, 01 Feb 2000 05:00:00 GMT
        Subject:
            C=GB, L=Cambridge, O=Zeus Technology, OU=Alice's machine
            CN=www.alice.zeus.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            Public Key:
                Modulus:
                    bb:c0:28:d0:86:73:02:c3:f6:04:72:23:b8:4a:f0:0e:35:
                    a8:5f:d8:74:41:3c:9c:6a:6e:93:57:94:6e:b4:ca:55:ca:
                    4f:81:cb:a7:39:bd:f0:47:02:ee:33:1f:ba:64:40:dd:30:
                    92:4c:02:8d:d3:95:34:d7:fb:d0:c1:37:09
                Exponent:
                    01:00:01
    Signature Algorithm:
        md5withRSAEncryption
    Signature:
        9c:30:8f:d6:6e:fe:7b:8f:d0:1d:3c:f2:c9:d5:be:3e:73:04:87:41:39:
        e1:df:fd:05:ad:f6:bb:d1:0f:ca:04:e5:65:4d:94:eb:16:27:a4:b4:73:
        15:25:c6:a0:a3:51:2c:9d:ed:fc:7b:fd:79:c8:36:97:ed:3b:d5:5d:a7:
        ee:d0:b7:2b:cf:48:e2:e9:07:f1:45:f5:f7:10:25:98:97:95:c0:b8:8d:
        d8:06:6a:3c:96:50:80:a5:99:9c:2e:84:df:6b:3f:da:43:88:ff:3a:95:
        6d:17:5d:c3:66:9f:a4:23:b0:c8:04:29:37:2e:d8:52:0c:6b:6c:4f:ac:
        d7:f3
```

The certificate contains 4 components:

- **Issuer**: This identifies the Certificate Authority who issues and signed this certificate. A client application (e.g. a web browser) will check its internal list of trusted Certificate Authorities. If it trusts this CA, it will extract the CA's public key from ithe CA's certificate (also stored internally) and use this to verify the digital signature on this certificate. In this case, the issuer the the Thawte Test CA Root.

- **Validity**: CA's typically issue certificates with a limited term of validity (1 to 3 years is common). This test certificate has a validity of 1 month.
- **Subject**: This is the owner of the certificate. The certificate identifies the subject's country (C), locality (L), organisation (O), organisational unit (OU) and common name (CN). In the case of website certificates, the common name is used to record the DNS name of the subject. For this certificate, the subject's primary identity is its common name '`www.alice.zeus.com`'.
- **Subject Public Key Info**: This contains the public key of the subject.

The certificate is signed with the private key of the issuing authority (the CA). In this case, the signature is calculated from an MD5 hash of the document (the 'message digest'). This message digest is then encrypted with the signer's private key.

A client can verify the signature by decrypting the signature with the signer's public key, and comparing the result with its own calculation of the message digest. If the two match, the client can be confident that the document was signed by the signer.

Clients who possess and trust the public certificate of the signing CA can hence establish trust of the certificate.

3.3 Chains of Trust

Web browsers and other client applications are distributed with a preconfigured list of trusted CAs and their public certificates. These CAs are often referred to as 'root CAs'.

Other CAs can obtain signing certificates signed by one of these trusted root CAs. They can use these certificates to sign public website certificates. In this case, a chain of trust begins to grow and certificate signing can be delegated to reliable subordinate CAs.

For example, trusted root CA '**C**' decides that organisation '**B**' can operate as a legitimate certificate agency and signs their a public certificate. '**B**' can then operate and sign certificate requests.

Suppose '**B**' signs Alice's certificate '**A**'. A client cannot authenticate Alice's certificate '**A**', because the client does not have a prior trust of the signing certificate '**B**'.

However, if Alice presents both her own certificate '**A**' and her issuer's certificate '**B**', the client can authenticate her through the chain of trust that was established from '**A**' to '**B**' to '**C**' (the trusted root CA). The concatenation of '**A**' and '**B**' is referred to as a *certificate chain*.

In a chained certificate scenario, the root CA who issued the subordinate CA's certificate takes on a degree of liability for the subordinate's actions and security, and in many cases (e.g. public website authentication) is legally liable. For example, Entrust operate as a subordinate CA to Thawte.

The root CA will impose strict requirements on the operating procedures of its subordinate CAs and will require regular independent security audits. It may also require extensive liability insurance from the subordinate CA.

Certificate chaining is commonly used in large corporate intranets where it can be leveraged in a secure manner to delegate signing authority to subordinate entities. It is also sometimes used in internal CA systems where the liability issues are easily satisfied. For example, Verisign's public Global Server ID programme uses a subordinate certificate descended from one of Verisign's public root certificates.

3.4 Other Authentication Issues

Subverting SSL sites

A correctly operating SSL secure site depends on a number of correctly configured and trusted services. The PKI discussed above is only one significant part of this.

Clients must ensure that their CA database is not corrupt and has not been subverted. It is possible to add or remove root certificates from a web browser CA database, and an attacker could subvert the security offered by loading his own CA root certificate into a user's browser database. If an attacker can achieve this, he is in a position where he can distribute his own certificates. These certificates will be automatically trusted by the subverted clients.

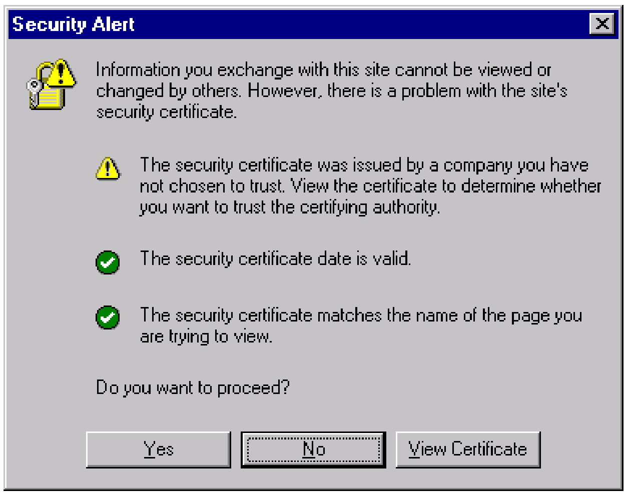
The design of SSL and the PKI places the onus on the end user to verify the correctness of his or her set of trusted certificates, and assumes that the client's local environment is secure.

A conceptually simple way to compromise an SSL site is for an attacker to subvert the DNS records for the real site so that they point to his own impersonating site. DNS records are regularly exchanged, updated and cached between co-operating DNS servers, and a number of exploits based on this technique have been reported. The attacker needs to obtain a valid public certificate for his fake site using the same common name (DNS) value as the site he is impersonating, but the certificate does not need to have been obtained from the same certificate authority.

Self-signed Certificates

Public Certificates can be signed by any entity. Obviously, only certificates that have been signed by recognised Certificate Authorities will be automatically accepted.

However, many client applications (including all web browsers) provide the ability to manually accept certificates on a session by session and site by site basis. Although not suitable for a public access web service, this technique is often used in an internal or development environment where authentication can either be established by other means, or simply is not significant for the website in question.


Security Alert

Information you exchange with this site cannot be viewed or changed by others. However, there is a problem with the site's security certificate.

The security certificate was issued by a company you have not chosen to trust. View the certificate to determine whether you want to trust the certifying authority.

The security certificate date is valid.

The security certificate matches the name of the page you are trying to view.

Do you want to proceed?

Dialog displayed by Internet Explorer when encountering an SSL site that is not certified by a recognised authority

It is common to 'self-sign' certificates. In this case, the issuer (signer) of the certificate is the same as the subject - in effect, the subject alone is vouching that they are who they say they are! This is the easiest way to create a public certificate in a stand-alone system, and many SSL-based products (such as the Zeus webserver) provide this facility.

All root CA certificates are self signed. This is required as they lie at the root of the certification and trust hierarchy and have no higher trusted entity to vouch for their identity. It illustrates of the degree of trust that must be placed in a root CA.

4 SSL Encryption

When a client (web browser) connects to an SSL-enabled server (web server), a 'handshake' procedure is undertaken to establish trust and encryption parameters.

During the handshake, the web server sends its public certificate to the client. The client can optionally decide to abort the connection as a result of authentication decisions make on the basis of the certificate, as described above.

If the client wishes to continue the connection, the next stage in the SSL handshake is to establish an encryption key. This key is used to encrypt the data sent over the SSL encryption layer.

4.1 Bulk Cipher Methods

We have already established that the public/private key might be suitable for safely encrypting data from the client to the server. However, public/private key methods are in general not suitable for a general purpose encryption methods.

For example, the RSA public/private key method can only encrypt blocks of data which are 11 bytes less than the key size. Each decryption operation involves complex mathematics - a typical high performance single processor server machine might be able to perform several tens or hundreds such decryptions per second with 1024 bit keys. Such a machine could only maintain an encryption rate of perhaps 10 KB/second. This throughput is extremely poor.

For this reason, SSL uses a different encryption method for encrypting the traffic across the SSL connection. A bulk cipher (encryption method) is used. Bulk ciphers are typically symmetric - both ends of the channel use the same keys to encrypt and decrypt the data - and very fast, so they are suitable for encrypting large quantities of data.

The following bulk ciphers algorithms are commonly used by SSL:

Cipher Identifier	*Type*	*Exportable ?*	*Key Source Size**	*Cipher Key Size**
`NULL`	Stream	Yes	0	0
`IDEA_CBC`	Block	No	128	128
`RC2_CBC_40`	Block	Yes	40	128
`RC4_40`	Stream	Yes	40	128
`RC4_56`	Stream	Yes	56	128
`RC4`	Stream	No	128	128
`DES40_CBC`	Block	Yes	40	56
`DES_CBC`	Block	No	56	56
`3DES_EDE_CBC`	Block	No	168	168

* Sizes measured in *bits*.

Of these ciphers, RC4 is by far the most widely deployed, and is seen in 40 and 56 bit more for export strength ciphers, and 128 bit more for full strength (US domestic) ciphers.

The two endpoints in the SSL conversation must agree on a pair of encryption keys for use with the bulk encryption algorithm. These keys must be established in a secure manner, so that an eavesdropper cannot determine its value. The client used the first key of the pair to encrypt the data it writes; the server uses the second key to encrypt the data it writes. The receiving endpoint uses the matching key to decrypt the data ir receives.

The keys are constructed using a complex combination of hashing algorithms that is difficult to reverse engineer. This key construction process takes three sources of data:

- **Client.random:** 32 bytes of random data generated by the client and exchanged in the clear
- **Server.random**: 32 bytes of random data generated by the server and exchanges in the clear
- **PreMasterSecret**: 48 bytes of random data generated by the client and public-key-encrypted before being sent to the server

The server and the client generate a 'MasterSecret' from this data using the complex hashing procedure described above. The client and the server both derive the same MasterSecret, but an eavesdropper who is snooping on the SSL conversation so far cannot derive the secret because he could not decrypt the public-key-encrypted PreMasterSecret chosen by the client.

Finally, the client and the server extract the correct number of bits from the generated MasterSecret to generate their bulk encryption keys. This number of bits used is listed in the table above (*Key Source Size*).

If the Key Source Size is less than the size of the key that the cipher algorithm requires (the *Cipher Key Size*), both endpoints will perform a final hash operation to expand the key to the correct size. In this case, the effective key size is less than the actual key size, making a brute-force attack on de-cyphering the data more feasible. This was a legal requirement of US export laws (such ciphers are branded 'export-strength ciphers'), and although these export restrictions have now been relaxed or repealed, there are still a large number of export-only browsers in common use today.

It is important to distinguish between the different keys used in an SSL transaction:

- **Public/Private Key Pair**:

  An RSA Public/Private key pair is most commonly used to exchange the data required to generate the bulk encryption key. The size (strength) of RSA keys is measured in bits - typically either 512 or 1024 bits in size.

- **Bulk Encryption Key**:

  The bulk encryption key is used to encrypt the application layer data in the SSL protected channel. In the case of the RC4 bulk encryption algorithm, this key is 128 bits in size. However, its effective size may be lower (40 or 56 bits if of export strength).

When the strength of an SSL connection is specified, it is usually specified in terms of the effective size of the bulk encryption key.

4.2 Message Authentication Codes

Every encrypted message exchanged between the client and the server is protected from tampering with a MAC (*Message Authentication Code*). The MAC is calculated by hashing the body of the message (before encryption) along with some additional deterministic data (sequence number, length, padding) and a *MACWriteSecret*.

The MACWriteSecret is derived from the shared MasterSecret and is unknown by an eavesdropper. Consequently, it is not possible (without extensive brute force investigation) for an attacker to generate a valid MAC for given data, and as a result, it is not practically possible for an attacker with access to the communication channel to modify the data in transit without detection.

The MAC provides the reliability of the message channel against tampering that many secure applications require.

The hash function used to calculate the MAC is either MD5 or SHA1.

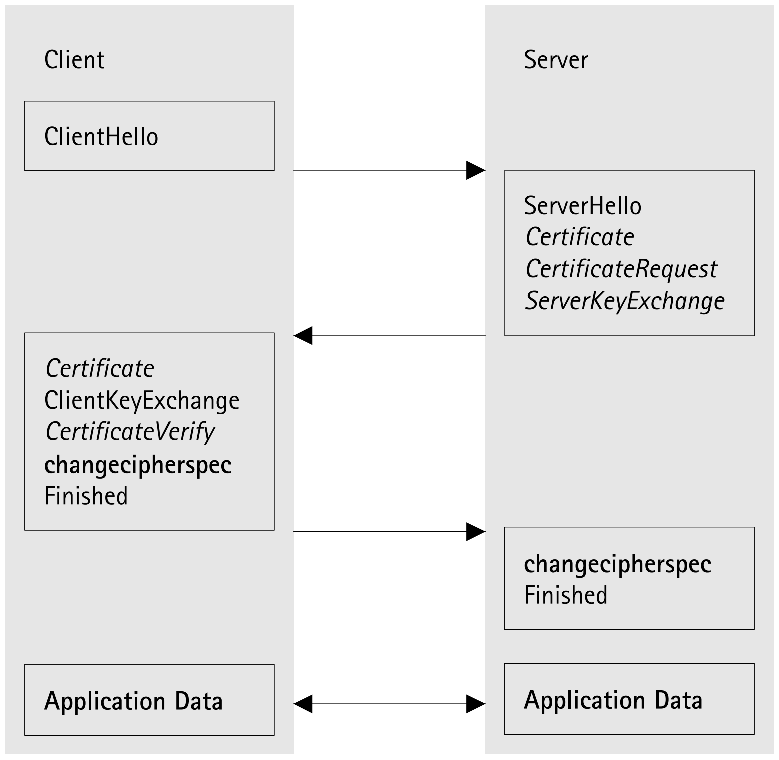
5 SSL Transactions

This chapter will tie together the information presented in previous chapters about the SSL authentication measures and the encryption and MAC layers.

5.1 The SSL Handshake

A SSL transaction begins with a handshake operation. In this handshake, the client and server agree on an encryption and MAC method, the server presents its public certificate and a common shared secret (the MasterSecret) is determined in a secure manner.

The client and the server then proceed with the SSL-protected application layer conversation.



SSL Handshake protocol

Optional or content-dependent Handshake Messages are shown in *italic* type.

The following messages are exchanged in the SSL handshake:

- **ClientHello**:

  This message initiates the SSL handshake. In this message, the client:

  - informs the server the highest version of SSL it can communicate with;
  - provides 32 bytes of random data (Client.random) for later use;
  - optionally suggests a session ID if it wishes to re-use a previous SSL session;
  - lists the cipher suites (key exchange, bulk cipher, MAC algorithm tuples) it supports, in order of preference.

- **ServerHello**:

  The server replies with a ServerHello message. In this message, the server:

  - nominates the version of SSL that will be used in this session;
  - provides 32 bytes of random data (Server.random) for later use;
  - provides an opaque session id so that this session can be re-used;
  - chooses the cipher suite to be used in the session.

- **Certificate**:

  If the server possesses a public certificate of the type chosen in the cipher suite, it passes this to the client.

  The client may also use this message type to send its own certificate to the server if requested. If this happens, the client also sends a CertificateVerify message where it encrypts some shared data with its private key; the server can use this message to verify that the client posses the corresponding private key.

- **ServerKeyExchange**:

  If the server's certificate cannot be used for encrypting data (it is not present, or export restrictions forbid the use of public keys greater than 512 bits for encryption purposes), the server generates a temporary public/private key pair and uses this message to pass the public portion of the key pair to the client.

- **CertificateRequest**:

  The server can optionally send a message requesting the client to send its own identifying public certificate (if it possesses one). The server may be configured to reject connections from clients that do not present suitably authorised certificate identification.

- **ClientKeyExchange**:

  This message is sent by the client. It contains the encrypted PreMasterSecret which is ultimately used to generate the encryption and MAC parameters.

- **changecipherspec**:

  This is a special out-of-band message. Either party sends a changecipherspec message before it activates the pending encryption method and MAC algorithm (the *cipher suite*).

- **Finished**:

  The Finished message completes the SSL handshake operation. Once sent, the sender can then begin transmitting application level data, encrypted and MACed with the current in-force cipher suite.

  The Finished message includes a hash of all previous handshake messages. The receiver can compare this hash with its own knowledge and hence detect errors or inconsistencies in the handshake procedure.

At any point, either party in the handshake is free to abort the connection if it receives an unexpected message, or one it cannot handle.

5.2 Session Keys

The SSL handshake is an expensive operation. In particular, the decryption of the ClientKeyExchange performed by the server side requires an Private Key decrypt operation which is computationally expensive. In addition, the network latency overhead of exchanging the handshake messages impacts on the perceived SSL performance.

For this reason, when an SSL handshake is performed, the SSL server assigns an opaque *session key*. The server records the session parameters (cipher suite, client certificates, MasterSecret) in its session cache.

A client can reconnect to the server and supply a previously used session key. In this case, the client and server can reuse the session parameters and therefore do not need to complete a full SSL handshake operation.

6 Export Restrictions on SSL Technology

6.1 Overview

US export restrictions used to prohibit the export of high-strength encryption software to the rest of the world, on the grounds that they placed a threat to US national security. 128-bit encryption technologies were classified as munitions.

For this reason, weaker 40-bit ciphers were developed and software licensed for export from the US had to use ciphers that were no stronger than these. Consequently, many web browsers were distributed in two forms: export strength (40 bit) and domestic strength (128 bit). Domestic strength browsers could not legally be used outside the United States.

In January 2000, the export restrictions were relaxed and 56 bit ciphers were licensed for 'foreign' use.

As a result of heavy industry pressure and increased computing power making the breaking of the weak export-strength ciphers more feasible, in April 2000, these export restrictions were further relaxed and web browser manufacturers were licensed to export full 128-bit strength web clients to all but specific US-embargoed destinations.

Web server software has typically been free from these export restrictions, to allow US nationals to access foreign-located websites at full domestic strength.

The Zeus Webserver supports all commonly used ciphers, both domestic and export strength, and has always been free from US export restrictions.

There are still a large number of export browsers in circulation, but it is expected that this number will decrease with time.

6.2 Global Server Ids

Many international internet users expressed concern about the security of 40 bit SSL transactions The US responded by allowing two certificate agencies to issue special 'step-up' certificates to recognised financial institutions and some e-Commerce sites.

Verisign branded the certificates 'Global Server IDs'; Thawte branded them as 'SuperCerts'.

When an export strength web client connects to an SSL-enabled website, it initally nominates a set of export strength ciphers for the server to select from for the SSL session. If, during the handshake, the client receives a trusted 'step-up' type certificate, it has the option of reconnecting to the site, nominating full-strength ciphers for the transaction. As a result, the technology lets certain websites with the step-up enabled certificates allow clients to connect at full 128 bit strength, regardless of whether they are export or domestic strength.

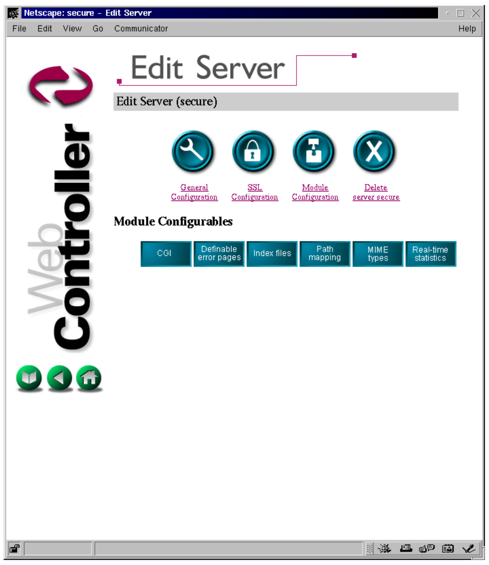
The export restrictions on step-up certificates have now been lifted, and although the technology is no longer necessary, many sites are now applying for these certificates in order to ensure that users of older export-only web clients can connect at full encryption strength.

7 Configuring the Zeus Webserver for SSL

7.1 SSL Quick Setup

Zeus Server allows each of your virtual servers to have their own SSL certificates. This allows multiple secure sites to run on the same Zeus server. Each virtual server requires its own public and private certificates for secure communication.

Secure certificates are added to a virtual server by clicking on the SSL Configuration link from the Edit Server page:


Netscape: secure - Edit Server
File Edit View Go Communicator Help
Web Controller

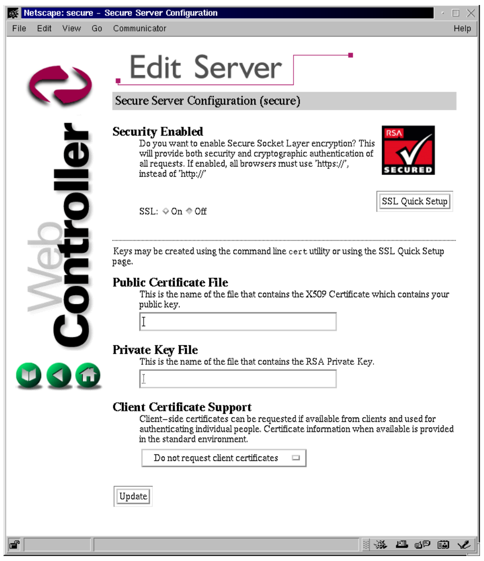
Edit Server

Edit Server (secure)

General Configuration
SSL Configuration
Module Configuration
Delete server secure

Module Configurables

CGI | Definable error pages | Index files | Path mapping | MIME types | Real-time statistics


Netscape: secure - Secure Server Configuration
File Edit View Go Communicator Help
Web Controller
RSA SECURED

Edit Server

Secure Server Configuration (secure)

Security Enabled
Do you want to enable Secure Socket Layer encryption? This will provide both security and cryptographic authentication of all requests. If enabled, all browsers must use 'https://', instead of 'http://'

SSL Quick Setup

SSL: On Off

Keys may be created using the command line `cert` utility or using the SSL Quick Setup page.

Public Certificate File
This is the name of the file that contains the X509 Certificate which contains your public key.

Private Key File
This is the name of the file that contains the RSA Private Key.

Client Certificate Support
Client–side certificates can be requested if available from clients and used for authenticating individual people. Certificate information when available is provided in the standard environment.

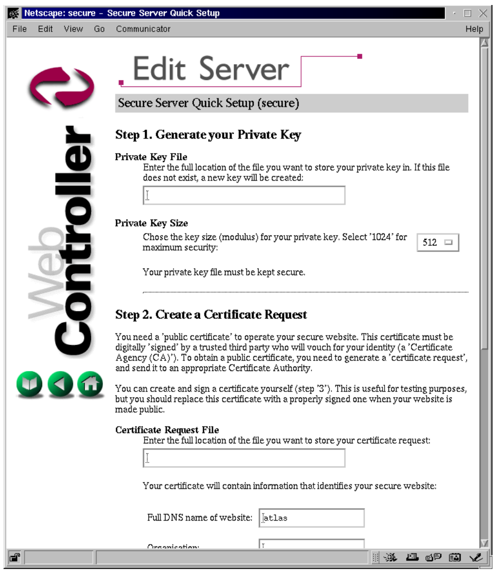
Do not request client certificates

Update

Zeus Admin Server: Edit Server

Zeus Admin Server: SSL Configuration

The easiest way to configure SSL on your virtual server is to use the 'SSL Quick Setup' wizard. Click the 'SSL Quick Setup' button and fill in the form:


Netscape: secure - Secure Server Quick Setup
File Edit View Go Communicator Help
Web Controller

Edit Server

Secure Server Quick Setup (secure)

Step 1. Generate your Private Key

Private Key File
Enter the full location of the file you want to store your private key in. If this file does not exist, a new key will be created:

Private Key Size
Chose the key size (modulus) for your private key. Select '1024' for maximum security: 512

Your private key file must be kept secure.

Step 2. Create a Certificate Request

You need a 'public certificate' to operate your secure website. This certificate must be digitally 'signed' by a trusted third party who will vouch for your identity (a 'Certificate Agency (CA)'). To obtain a public certificate, you need to generate a 'certificate request', and send it to an appropriate Certificate Authority.

You can create and sign a certificate yourself (step '3'). This is useful for testing purposes, but you should replace this certificate with a properly signed one when your website is made public.

Certificate Request File
Enter the full location of the file you want to store your certificate request:

Your certificate will contain information that identifies your secure website:

Full DNS name of website: atlas

Organisation:

Zeus Admin Server: SSL Quick Setup

- **private key**: file name and key size (recommend 1024 bit key size)
- **certificate request parameters**: file name and identification details
- **public certificate**: file name

Click on 'Setup Server'. This will generate a private key, certificate request and self-signed public certificate. It will store the certificate request in the named file, and also display it on the screen so you can copy and paste it into your CA's online application form.

If your website is running on the default non-secure port (80), you will need to change it to use port 443, which is the default port for SSL:

1. Click on 'General Configuration'.
2. In the 'Server Address' form, change the port number from '80' to '443'.
3. Click 'Update'.

Stop and start your website (using the 'traffic lights').

You website will now be operating in SSL mode. It will use an interim self-signed certificate, and your website users will be prompted to accept this certificate each time they access your site.

Obtaining a certificate from a CA:

You can now apply for a properly signed certificate from a Certificate Authority such as Verisign. You will need to provide the Certificate Request details (the CSR) during the application process.

The Certificate Authority will supply your signed certificate by email or by delivery over the web.

In many cases, you can just replace the contents of the public certificate file (the self signed cert) with the new one. Alternatively, you can use the 'SSL Configuration' form to change the filename that webserver uses for the public certificate.

However, your CA may require that you create a certificate chain to use their certificate. In this case, you will receive two certificates from the CA in response to your signing request. One of these will be your public certificate, and the other an intermediate certificate. Append the intermediate certificate onto your public certificate to create your certificate chain:

```
$ cat public.cert intermediate.cert > chained.cert
```

Replace your self-signed public certificate file with your new certificate chain file.

Restart your website for the changes to take effect.

You can also use the *Secure Server Configuration* page to configure your SSL settings:

1. Use the enabled checkbox to turn on SSL. This provides an easy means of turning SSL on and off for the server.
2. The Public Certificate File should be a signed X509 certificate from your Certificate Authority. Enter the full pathname of the file from the server's root directory..
3. The Private Certificate File should be your private key from which you generated the certificate request. Enter the full pathname of the file from the servers root

directory. Under no circumstances should the private key be stored in the document root of the virtual server.

4. Click on the Update button to commit the configuration changes and return to the *Edit Server* page. You will need to restart the virtual server from the *Controller* to allow the changed to take effect.

7.2 The `cert` Tool

You can also configure your SSL server manually. You can use the `cert` tool (`$ZEUSHOME/admin/bin/cert`) or an equivalent tool like `openssl`[vi] to generate the keys and certificates.

Generating a private key:

```
$ $ZEUSHOME/admin/bin/cert -new -type private -keysize 1024 -out private.key
Generating keys, this may take a few seconds
Your new private key has been written to 'private.key'
```

Generating a certificate request:

```
$ $ZEUSHOME/admin/bin/cert -new -type request -key private.key -out cert.csr

The following information is required to make up the certificate.
Optional fields can be left blank by entering a '.'

Country: GB
State/Province (optional outside US): .
Locality (town/city): Cambridge
Organisation: Zeus
Organisational Unit (optional): .
Common Name (full DNS name of the machine): www.zeus.com

Your certificate request has been written to the file 'cert.csr'.
```

Generating a public certificate:

```
$ $ZEUSHOME/admin/bin/cert -new -type public -key private.key -out public.cert

The following information is required to make up the certificate.
Optional fields can be left blank by entering a '.'

Country: GB
State/Province (optional outside US): .
Locality (town/city): Cambridge
Organisation: Zeus
Organisational Unit (optional): .
Common Name (full DNS name of the machine): www.zeus.com

Your new public certificate has been written to the file 'public.cert'.
```

Displaying the public certificate:

```
$ $ZEUSHOME/admin/bin/cert -in public.cert -text

    X509 Certificate:
        Certificate Info:
            Version: 00
            Serial Number:         00
            Signature Algorithm: md5withRSAEncryption
            Issuer:
                C=UK, L=Cambridge, O=Zeus, CN=atlas
            Validity:
                Not Before: Mon, 15 May 2000 17:40:11 GMT
                Not After:  Tue, 15 May 2001 17:40:11 GMT
            Subject:
                C=GB, L=Cambridge, O=Zeus, CN=www.zeus.com
            Subject Public Key Info:
                Public Key Algorithm: rsaEncryption
                Public Key:
                    Modulus:
                        c4:84:c7:63:36:e0:d4:52:fb:41:44:31:38:9b:91:5e:62:
                        ....
                    Exponent:
                        01:00:01
        Signature Algorithm:
            md5withRSAEncryption
        Signature:
            29:eb:c2:62:73:b8:b7:82:94:33:1f:da:9b:83:39:8b:75:1c:47:f5:41:
            ....
```

Verifying Key/Certificate pairs:

```
$ $ZEUSHOME/admin/bin/cert -check -key private.key -in public.cert

private and public key are a valid pair
```

7.3 Tuning your Zeus SSL server for performance

The bottleneck for SSL-encrypted transactions is the initial key exchange (the 'SSL accept'). This key exchange involves an RSA decrypt, and it establishes a key for the bulk cipher used to encrypt the rest of the session. It also establishes a unique session id which the client and server use to cache the established session parameters.

All the HTTP requests/responses are encrypted by a bulk cipher, which is very fast. You'll typically see about 80-90% performance (throughput) compared to un-encrypted sessions.

The key (if you'll excuse the pun) to fast SSL performance is managing your session ids. If your webserver has cached the session id, it won't need to force another expensive SSL accept.

Zeus uses a two-level caching system for the session ids. Most recently used session ids are cached in memory. All session ids (including the in-memory cached values) are cached to a second level disk cache which can be shared between multiple instances of the webserver.

- **Level 1 (memory)**: tuneables in `$ZEUSHOME/web/global.cfg`:
  - `tuning!ssl_sessioncache_size` - max no. of entries in your L1 cache. Defaults to 199. You can try a large (prime) number, but watch your memory usage. Consider the second level cache as an alternative.

- **Level 2 (disk)**: tuneables in `$ZEUSHOME/web/global.cfg`:
  - `tuning!ssl_diskcache` - defaults to 'no' for one zeus.web child, 'yes' if you're running multiple zeus.web children (eg on a multiprocessor system). The cache is created in `$ZEUSHOME/web/ssl_cache`.

- `tuning!ssl_sessioncache_expiry` - defaults to 24*60*60 (24 hours).

The SSL keepalive feature allows a client to hold an SSL / TCP/IP connection open for future use. This reduces the number of expensive TCP/IP and SSL connects in an SSL session. By default, the `tuning!ssl_keepalive` tunable is set to '`no`', but setting it to '`yes`' provides a significant performance boost. However, some older

Microsoft clients (early versions of IE 4) don't implement SSL keepalive correctly, so you may wish to keep this feature disabled.

7.4 Using SSL with Clustering or Load Balancing solutions

The real killer comes when you need to scale your webserver installation to more than one machine. If you use round robin DNS or a non-SSL-aware load balancer, you'll find your clients getting session id cache misses and having to re-establish their SSL sessions on most of their connects. In this case, horizontally scaling your installation will severely **decrease** your performance!

Zeus clustering solves this problem. You can enable the second level SSL disk cache and NFS share it across your clustered machines.

If you have a mixed environment of webservers, or you need a high performance, fault tolerant load balancing solution, you'll need Zeus's load balancer product. This load balancer (typically a pair of machines) sits in front of a webserver farm and distributes requests to the back end webservers. It is SSL aware, caches SSL session ids, and will whenever possible route an SSL connection to the backend webserver that established the connection.

7.5 Client-Side Authentication with the Zeus Webserver

The SSL specification provides a means for the connecting clients to be sure of the identity of the website they are connecting to. Client-side certificates enable the webserver to similarly authenticate the connecting clients.

Many web clients can be configured with client-side certificates. These certificates can be used to identify the client. You can configure your Zeus server to request or require that connecting clients supply their certificates:

1. Configure Zeus to request or require client certificates using the *SSL Configuration* page.
2. Place the root certificates you are prepared to accept in a directory named `$ZEUSHOME/etc/CAs` in each installation of your webserver.

If you **require** a client-side certificate and the client does not supply one, the SSL connection is rejected (terminated).

If you **request** or **require** a client-side certificate and the client does supply one, then the certificate is checked against the root certificates in the `$ZEUSHOME/etc/CAs` directory. If it has not been issued by one of these CAs or if the certificate has expired, the connection is rejected.

If the connection is accepted, and the client has supplied a certificate, the certificate data is supplied via the supported dynamic content interfaces. For example, you could construct a FastCGI authorizer to perform further checks on the certificate (against a Certificate Revocation List, for example). The variables passed to a CGI-like process are as follows:

• **SSL_CLIENT_CN**	Client Common Name
• **SSL_CLIENT_EMAIL**	Client email (optional)
• **SSL_CLIENT_OU**	Client Organisational Unit (optional)
• **SSL_CLIENT_O**	Client Organisation
• **SSL_CLIENT_L**	Client Locality (e.g., Town)
• **SSL_CLIENT_SP**	Client US State (optional)
• **SSL_CLIENT_C**	Client Country
• **SSL_CLIENT_ICN**	Issuer Common Name
• **SSL_CLIENT_IEMAIL**	Issuer email (optional)
• **SSL_CLIENT_IOU**	Issuer Organisational Unit (optional)

SSL_CLIENT_IO	Issuer Organisation
SSL_CLIENT_IL	Issuer Locality (e.g., Town)
SSL_CLIENT_ISP	Issuer US State (optional)
SSL_CLIENT_IC	Issuer Country
CLIENT_CERT	PEM encoded Client Certificate
SSL_CLIENT_SERIAL	Certificate Serial Number

The following values (edited for reasons of privacy) were supplied by a client certificate from a popular certificate authority:

- `CLIENT_CERT=MII0BAQQFADCBlDELMAkGA1UEBhMCWkE ...`
- `SSL_CLIENT_SERIAL=237533`
- `SSL_CLIENT_C=`
- `SSL_CLIENT_CN=Joe User`
- `SSL_CLIENT_EMAIL=joe@user.com`
- `SSL_CLIENT_L=`
- `SSL_CLIENT_O=`
- `SSL_CLIENT_OU=`
- `SSL_CLIENT_SP=`
- `SSL_CLIENT_IC=US`
- `SSL_CLIENT_ICN=MyCA (personal public certification)`
- `SSL_CLIENT_IEMAIL=`
- `SSL_CLIENT_IL=Memphis`
- `SSL_CLIENT_IO=MyCA`
- `SSL_CLIENT_IOU=Certificate Services`
- `SSL_CLIENT_ISP=Tennesse`

8 Useful SSL Resources

8.1 Software

cert

cert (documented above) is a key and certificate generation tool distributed with Zeus.

OpenSSL

OpenSSL (see www.openssl.org) is a powerful open source implementation of various SSL technologies. It includes tools for generating and managing keys, querying SSL servers and encrypting and decrypting data. It is an extremely useful tool for debugging SSL-related problems.

8.2 Web Resources

SSL version 3

Netscape hold the specification for SSL version 3 at home.netscape.com/eng/ssl3/.

Public Key Cryptography Standards (PKCS)

RSA labs hold the PKCS specifications in their PKCS standards center at www.rsasecurity.com/rsalabs/pkcs/. PKCS specifies many of the underlying technologies of SSL, including the encryption and certificate formats.

i RSA Crypto Challenge Sets New Security Benchmark (http://www.rsasecurity.com/news/pr/990826-2.html)
ii RSA Laboratories' Bulletin #13 (http://www.rsasecurity.com/rsalabs/bulletins)
iii Verisign (http://www.verisign.com)
iv Thawte (http://www.thawte.com)
v Entrust (http://www.entrust.com)
vi OpenSSL (http://www.openssl.org)
vii Thawte Test Server Certificates (https://www.thawte.com/cgi/server/test.exe)