SETUP CERTIFICATION AUTHORITY

1. **Creazione cartella CA**
   ```
   mkdir CA
   cd CA/
   ```

2. **Copia del file di configurazione standard**
   ```
   cp /etc/ssl/openssl.cnf .
   ```

3. **Creazione della struttura di directory prevista dal file di configurazione**
   ```
   mkdir demoCA
   cd demoCA/
   mkdir certs
   mkdir crl
   mkdir newcerts
   mkdir private
   touch index.txt
   ```

4. **Modifica dei file contenenti i numeri di serie**
   ```
   echo "01" > serial
   echo "01" > crlnumber
   ```

5. **Creazione del certificato della CA**
   ```
   openssl req -x509 -config openssl.cnf -days 365 -out demoCA/cacert.pem
   -keyout demoCA/private/cakey.pem -new -nodes
   ```

6. **Visualizzazione del certificato appena creato**
   ```
   openssl x509 -text -in cacert.pem | less
   ```

7. **Creazione cartella che conterrà le richieste di certificati degli utenti e le loro chiavi**
   ```
   cd ..
   mkdir utenti
   cd utenti/
   ```

8. **UTENTE: creazione della richiesta di certificato**
   ```
   openssl req -new -out user1req.pem -keyout user1key.pem -days 365
   -config ../openssl.cnf -newkey rsa:2048 -nodes
   ```

9. **Creazione del certificato per l'utente**
   ```
   cd ..
   openssl ca -config openssl.cnf -policy policy_anything -infiles
   utenti/user1req.pem
   cd demoCA/newcerts/
   ```

SETUP CERTIFICATION AUTHORITY SECONDO LIVELLO – CONFIGURAZIONE FIREFOX

1. **Creazione cartella CA**
   ```
   mkdir CA
   cd CA/
   ```

2. **Copia del file di configurazione standard**
   ```
   cp /etc/ssl/openssl.cnf .
   ```

3. **Creazione della struttura di directory prevista dal file di configurazione**
   ```
   mkdir demoCA
   cd demoCA/
   mkdir certs
   mkdir crl
   mkdir newcerts
   mkdir private
   touch index.txt
   ```

4. **Modifica dei file contenenti i numeri di serie**
   ```
   echo "01" > serial
   echo "01" > crlnumber
   ```

5. **Creazione della richiesta di certificato firmato dalla CA**
   ```
   openssl req -new -out CAreq.pem -keyout private/cakey.pem -days 365
   -config ../openssl.cnf -newkey rsa:2048 -nodes
   ```

6. **Inviare alla Root CA la richiesta CAreq.pem**
7. **Aspettare che la Root CA ci consegni il certificato firmato (**`demoCA/cacert.pem`**)**

8. **Visualizzazione del certificato della nostra CA**
   ```
   openssl x509 -text -in cacert.pem | less
   ```

9. **Creazione cartella che conterrà le richieste di certificati degli utenti e le loro chiavi**
   ```
   cd ..
   mkdir users
   cd users/
   ```

10. **UTENTE: creazione della richiesta di certificato**
    ```
    openssl req -new -out user1req.pem -keyout user1key.pem -days 365
    -config ../openssl.cnf -newkey rsa:2048 -nodes
    ```

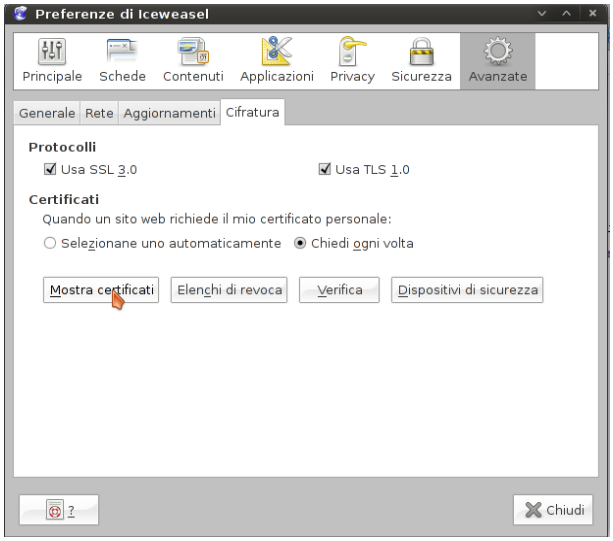
11. **Creazione del certificato per l'utente**
    ```
    cd ..
    openssl ca -config openssl.cnf -policy policy_anything -infiles
    users/user1req.pem
    ```

12. **Recuperare il certificato della Root CA (**`demoCA/cacert.pem`**)**
13. **Creazione di un file contenente il certificato della Root CA e il certificato della CA di secondo livello per esportazione in formato PKCS12**
    ```
    cd ..
    cat cacert.pem > cacacert.pem
    cat rootcacert.pem >> cacacert.pem
    ```

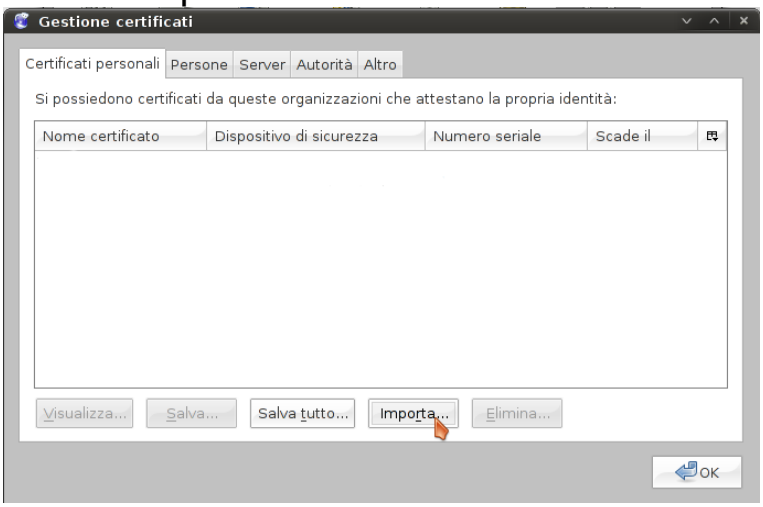
14. **Esportazione del certificato utente in formato PKCS12**

```
openssl pkcs12 -export -chain -CAfile cacacert.pem -inkey
../users/user1key.pem -name X5 -in newcerts/01.pem -out newcerts/user1.p12
```

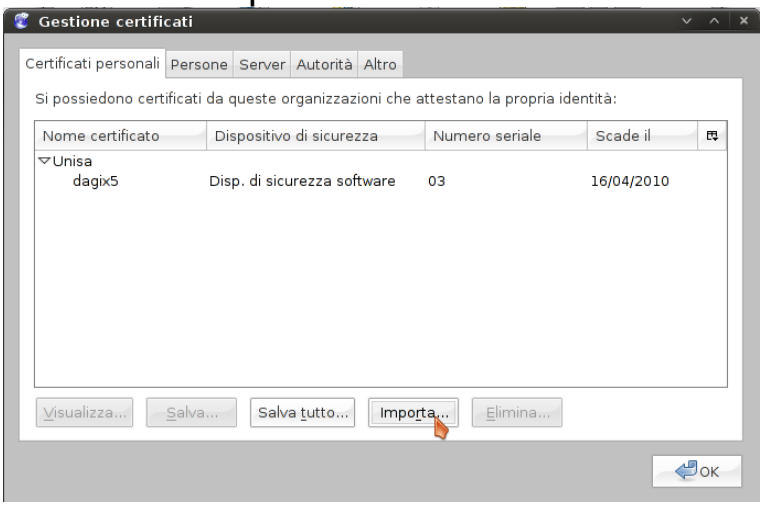
15. **Aprire Firefox: Modifica → Preferenze → Avanzate → Cifratura → Mostra Certificati**



16. **Certificati personali → Importa**



17. **Scegliere il file PKCS12 creato precedentemente**



ESECUZIONE APPLICAZIONI DI ESEMPIO

1. **Collegamento server web s_client**
   ```
   openssl s_client -connect shop.wind.it:443
   ```

2. **Collegamento s_server s_client**
   ```
   openssl s_server -accept 8443
   openssl s_client -connect localhost:8443
   ```

3. **Collegamento s_server s_client con certificati**
   ```
   cd CA1
   openssl s_server -accept 8443 -cert demoCA/cacert.pem -key demoCA/private/
   cakey.pem
   openssl s_client -connect localhost:8443 -CAfile demoCA/rootcacert.pem
   ```

4. **Collegamento s_server browser web**
   ```
   openssl s_server -accept 8443 -cert demoCA/cacert.pem -key demoCA/private/
   cakey.pem -WWW
   ```
   Aprire Firefox e collegarsi a https://localhost:8443

CONFIGURAZIONE APACHE2-SSL

0. **Cartella dei file di configurazione**
   ```
   cd /etc/apache2
   ```

1. **Creazione copia di backup del file di configurazione**
   ```
   cd sites-available
   cp default-ssl default-ssl.ok
   ```

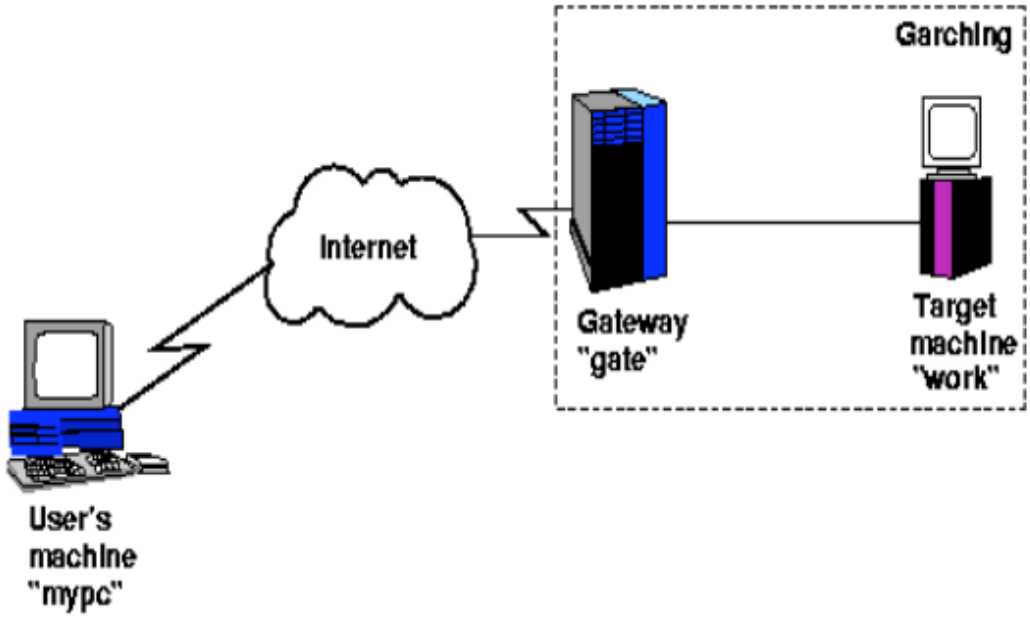
2. **Abilitazione della configurazione di SSL**
   ```
   ln -s default-ssl ../sites-enabled/001-default-ssl
   ```

3. **Se all'avvio di apache: `apache2: Could not reliably determine the server's fully qualified domain name, using 127.0.1.1 for ServerName`**
   ```
   sudo gedit /etc/apache2/apache2.conf
   ```
   aggiungere:
   ```
   ServerName "localhost"
   ```

4. **Modifiche da effettuare al file di configurazione di SSL**
   ```
   sudo gedit default-ssl
   ```
   modifiche:
   ```
   ...

   SSLEngine on

   SSLCertificateFile    /etc/apache2/CA/demoCA/newcerts/webserver.pem
   SSLCertificateKeyFile /etc/apache2/CA/users/webserverkey.pem

   SSLCACertificateFile /etc/apache2/CA/demoCA/cacacert.pem

   SSLCARevocationPath /etc/apache2/CA/demoCA/crl

   SSLVerifyClient require
   SSLVerifyDepth  10

   ...
   ```
   PS: il certificato webserver.pem contiene come Common Name "localhost", in quanto tale campo deve coincidere con l'indirizzo a cui ci si collega (https://localhost)

5. **Creazione pagina di esempio (richiede PHP)**
   ```
   cd /var/www
   gedit index.php

   <html>
         <head>
               <title> Apache2 SSL </title>
         </head>
         <body>
               <h2>Hi <?php print_r($_SERVER[SSL_CLIENT_S_DN_CN]); ?>,
   Apache2+SSL works!!</h2>
         </body>
   </html>
   ```

SSH TUNNELING



Scenario:

- nel nostro esempio "mypc" e "gate" sono la stessa macchina
- vogliamo collegarci al sito www.unisa.it ("work")creando un tunnel SSH tra "mypc" e "gate"

1. Eseguiamo il server SSH ("gate")
   `sudo /etc/init.d/ssh start`

2. Creiamo il tunnel
   `ssh -l umberto -L 7777:www.unisa.it:80 127.0.0.1`
   umberto è il login sulla macchina "gate"
   7777 è la porta locale di entrata del tunnel
   www.unisa.it è la macchina target
   80 è la porta target
   127.0.0.1 è la macchina "gate"

3. Verifichiamo che funzioni
   Colleghiamoci tramite il browser all'indirizzo http://127.0.0.1:7777 e dovrebbe *apparire* il sito www.unisa.it

Certificate Contents

A certificate associates a public key with the real identity of an individual, server, or other entity, known as the subject. As shown in Table 1, information about the subject includes identifying information (the distinguished name), and the public key. It also includes the identification and signature of the Certificate Authority that issued the certificate, and the period of time during which the certificate is valid. It may have additional information (or extensions) as well as administrative information for the Certificate Authority's use, such as a serial number.

Table 1: Certificate Information

Subject	Distinguished Name, Public Key
Issuer	Distinguished Name, Signature
Period of Validity	Not Before Date, Not After Date
Administrative Information	Version, Serial Number
Extended Information	Basic Constraints, Netscape Flags, etc.

A distinguished name is used to provide an identity in a specific context -- for instance, an individual might have a personal certificate as well as one for their identity as an employee. Distinguished names are defined by the X.509 standard [X509], which defines the fields, field names, and abbreviations used to refer to the fields (see Table 2).

Table 2: Distinguished Name Information

DN Field	Abbrev.	Description	Example
Common Name	CN	Name being certified	CN=Joe Average
Organization or Company	O	Name is associated with this organization	O=Snake Oil, Ltd.
Organizational Unit	OU	Name is associated with this organization unit, such as a department	OU=Research Institute
City/Locality	L	Name is located in this City	L=Snake City
State/Province	ST	Name is located in this State/Province	ST=Desert
Country	C	Name is located in this Country (ISO code)	C=XZ

A Certificate Authority may define a policy specifying which distinguished field names are optional, and which are required. It may also place requirements upon the field contents, as may users of certificates. As an example, a Netscape browser requires that the Common Name for a certificate representing a server has a name which matches a wildcard pattern for the domain name of that server, such as `*.snakeoil.com`.

The binary format of a certificate is defined using the ASN.1 notation [X208] [PKCS]. This notation defines how to specify the contents, and encoding rules define how this information is translated into binary form. The binary encoding of the certificate is defined using Distinguished Encoding Rules (DER), which are based on the more general Basic Encoding Rules (BER). For those transmissions which cannot handle binary, the binary form may be translated into an ASCII form by using Base64 encoding [MIME]. This encoded version is called PEM encoded (the name comes from "Privacy Enhanced Mail"), when placed between begin and end delimiter lines as illustrated in the following example.

Example of a PEM-encoded certificate (snakeoil.crt)

```
-----BEGIN CERTIFICATE-----
MIIC7jCCAlegAwIBAgIBATANBgkqhkiG9w0BAQQFADCBqTELMAkGA1UEBhMCWFkx
FTATBgNVBAgTDFNuYWtlIER1c2VydDETMBEGA1UEBxMKU25ha2UgVG93bjEXMBUG
A1UEChMOU25ha2UgT2lsLCBMdGQxHjAcBgNVBAsTFUNlcnRpZmljYXRlIEF1dGhv
cml0eTEVMBMGA1UEAxMMU25ha2UgT2lsIENBMR4wHAYJKoZIhvcNAQkBFg9jYUBz
bmFrZW9pbC5kb20wHhcNOTgxMDIxMDg1ODM2WhcNOTkxMDIxMDg1ODM2WjCBpzEL
MAkGA1UEBhMCWFkxFTATBgNVBAgTDFNuYWtlIER1c2VydDETMBEGA1UEBxMKU25h
a2UgVG93bjEXMBUGA1UEChMOU25ha2UgT2lsLCBMdGQxFzAVBgNVBAsTDldlYnNl
cnZlciBUZWFtMRkwFwYDVQQDExB3d3cuc25ha2VvaWwuZG9tMR8wHQYJKoZIhvcN
AQkBFhB3d3dAc25ha2VvaWwuZG9tMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKB
gQDH9Ge/s2zcH+da+rPTx/DPRp3xGjHZ4GG6pCmvADIEtBtKBFAcZ64n+Dy7Np8b
vKR+yy5DGQiijsH1D/j8HlGE+q4TZ8OFk7BNBFazHxFbYI4OKMiCxdKzdif1yfaa
lWoANFlAzlSdbxeGVHoTOK+gT5w3UxwZKv2DLbCTzLZyPwIDAQABoyYwJDAPBgNV
HRMECDAGAQH/AgEAMBEGCWCGSAGG+EIBAQQEAwIAQDANBgkqhkiG9w0BAQQFAAOB
gQAZUIHAL4DO9oE6Lv2k56Gp38OBDuILvwLg1v1KL8mQR+KFjghCrtpqaztZqcDt
2q2Qoyu1CgSzHbEGmiOEsdkPfg6mpOpenssIFePYNI+/8u9HT4LuKMJX15hxBam7
dUHzICxBVC11nHyYGjDuAMhe396lYAn8bCld1/L4NMGBCQ==
-----END CERTIFICATE-----
```

Certificate Authorities

By first verifying the information in a certificate request before granting the certificate, the Certificate Authority assures the identity of the private key owner of a key-pair. For instance, if Alice requests a personal certificate, the Certificate Authority must first make sure that Alice really is the person the certificate request claims.

Certificate Chains

A Certificate Authority may also issue a certificate for another Certificate Authority. When examining a certificate, Alice may need to examine the certificate of the issuer, for each parent Certificate Authority, until reaching one which she has confidence in. She may decide to trust only certificates with a limited chain of issuers, to reduce her risk of a "bad" certificate in the chain.

Creating a Root-Level CA

As noted earlier, each certificate requires an issuer to assert the validity of the identity of the certificate subject, up to the top-level Certificate Authority (CA). This presents a problem: Since this is who vouches for the certificate of the top-level authority, which has no issuer? In this unique case, the certificate is "self-signed", so the issuer of the certificate is the same as the subject. As a result, one must exercise extra care in trusting a self-signed certificate. The wide publication of a public key by the root authority reduces the risk in trusting this key -- it would be obvious if someone else publicized a key claiming to be the authority. Browsers are preconfigured to trust well-known certificate authorities.

A number of companies, such as Thawte and VeriSign have established themselves as Certificate Authorities. These companies provide the following services:

- Verifying certificate requests
- Processing certificate requests
- Issuing and managing certificates

It is also possible to create your own Certificate Authority. Although risky in the Internet environment, it may be useful within an Intranet where the organization can easily verify the

identities of individuals and servers.

Certificate Management

Establishing a Certificate Authority is a responsibility which requires a solid administrative, technical, and management framework. Certificate Authorities not only issue certificates, they also manage them -- that is, they determine how long certificates are valid, they renew them, and they keep lists of certificates that have already been issued but are no longer valid (Certificate Revocation Lists, or CRLs). Say Alice is entitled to a certificate as an employee of a company. Say too, that the certificate needs to be revoked when Alice leaves the company. Since certificates are objects that get passed around, it is impossible to tell from the certificate alone that it has been revoked. When examining certificates for validity, therefore, it is necessary to contact the issuing Certificate Authority to check CRLs -- this is not usually an automated part of the process.

Note

If you use a Certificate Authority that is not configured into browsers by default, it is necessary to load the Certificate Authority certificate into the browser, enabling the browser to validate server certificates signed by that Certificate Authority. Doing so may be dangerous, since once loaded, the browser will accept all certificates signed by that Certificate Authority.

Secure Sockets Layer (SSL)

The Secure Sockets Layer protocol is a protocol layer which may be placed between a reliable connection-oriented network layer protocol (e.g. TCP/IP) and the application protocol layer (e.g. HTTP). SSL provides for secure communication between client and server by allowing mutual authentication, the use of digital signatures for integrity, and encryption for privacy.

The protocol is designed to support a range of choices for specific algorithms used for cryptography, digests, and signatures. This allows algorithm selection for specific servers to be made based on legal, export or other concerns, and also enables the protocol to take advantage of new algorithms. Choices are negotiated between client and server at the start of establishing a protocol session.

Table 4: Versions of the SSL protocol

Version	Source	Description	Browser Support
SSL v2.0	Vendor Standard (from Netscape Corp.) [SSL2]	First SSL protocol for which implementations exists	- NS Navigator 1.x/2.x - MS IE 3.x - Lynx/2.8+OpenSSL
SSL v3.0	Expired Internet Draft (from Netscape Corp.) [SSL3]	Revisions to prevent specific security attacks, add non-RSA ciphers, and support for certificate chains	- NS Navigator 2.x/3.x/4.x - MS IE 3.x/4.x -

			Lynx/2.8+OpenSSL
TLS v1.0	Proposed Internet Standard (from IETF) [TLS1]	Revision of SSL 3.0 to update the MAC layer to HMAC, add block padding for block ciphers, message order standardization and more alert messages.	- Lynx/2.8+OpenSSL

There are a number of versions of the SSL protocol, as shown in Table 4. As noted there, one of the benefits in SSL 3.0 is that it adds support of certificate chain loading. This feature allows a server to pass a server certificate along with issuer certificates to the browser. Chain loading also permits the browser to validate the server certificate, even if Certificate Authority certificates are not installed for the intermediate issuers, since they are included in the certificate chain. SSL 3.0 is the basis for the Transport Layer Security [TLS] protocol standard, currently in development by the Internet Engineering Task Force (IETF).

Session Establishment

The SSL session is established by following a handshake sequence between client and server, as shown in Figure 1. This sequence may vary, depending on whether the server is configured to provide a server certificate or request a client certificate. Though cases exist where additional handshake steps are required for management of cipher information, this article summarizes one common scenario: see the SSL specification for the full range of possibilities.

Note

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

Figure 1: Simplified SSL Handshake Sequence

The elements of the handshake sequence, as used by the client and server, are listed below:

1. Negotiate the Cipher Suite to be used during data transfer
2. Establish and share a session key between client and server
3. Optionally authenticate the server to the client
4. Optionally authenticate the client to the server

The first step, Cipher Suite Negotiation, allows the client and server to choose a Cipher Suite supportable by both of them. The SSL3.0 protocol specification defines 31 Cipher Suites. A Cipher Suite is defined by the following components:

- Key Exchange Method
- Cipher for Data Transfer
- Message Digest for creating the Message Authentication Code (MAC)

These three elements are described in the sections that follow.

Key Exchange Method

The key exchange method defines how the shared secret symmetric cryptography key used for application data transfer will be agreed upon by client and server. SSL 2.0 uses RSA key exchange only, while SSL 3.0 supports a choice of key exchange algorithms including the RSA key exchange when certificates are used, and Diffie-Hellman key exchange for exchanging keys without certificates and without prior communication between client and server.

One variable in the choice of key exchange methods is digital signatures -- whether or not to use them, and if so, what kind of signatures to use. Signing with a private key provides assurance against a man-in-the-middle-attack during the information exchange used in generating the shared key [AC96, p516].

Cipher for Data Transfer

SSL uses the conventional cryptography algorithm (symmetric cryptography) described earlier for encrypting messages in a session. There are nine choices, including the choice to perform no encryption:

- No encryption
- Stream Ciphers
    - RC4 with 40-bit keys
    - RC4 with 128-bit keys
- CBC Block Ciphers
    - RC2 with 40 bit key
    - DES with 40 bit key
    - DES with 56 bit key
    - Triple-DES with 168 bit key
    - Idea (128 bit key)
    - Fortezza (96 bit key)

Here "CBC" refers to Cipher Block Chaining, which means that a portion of the previously encrypted cipher text is used in the encryption of the current block. "DES" refers to the Data Encryption Standard [AC96, ch12], which has a number of variants (including DES40 and 3DES_EDE). "Idea" is one of the best and cryptographically strongest available algorithms, and "RC2" is a proprietary algorithm from RSA DSI [AC96, ch13].

Digest Function

The choice of digest function determines how a digest is created from a record unit. SSL supports the following:

- No digest (Null choice)
- MD5, a 128-bit hash
- Secure Hash Algorithm (SHA-1), a 160-bit hash

The message digest is used to create a Message Authentication Code (MAC) which is encrypted with the message to provide integrity and to prevent against replay attacks.

Handshake Sequence Protocol

The handshake sequence uses three protocols:

- The SSL Handshake Protocol for performing the client and server SSL session establishment.
- The SSL Change Cipher Spec Protocol for actually establishing agreement on the Cipher Suite for the session.
- The SSL Alert Protocol for conveying SSL error messages between client and server.

These protocols, as well as application protocol data, are encapsulated in the SSL Record Protocol, as shown in Figure 2. An encapsulated protocol is transferred as data by the lower layer protocol, which does not examine the data. The encapsulated protocol has no knowledge of the underlying protocol.

Figure 2: SSL Protocol Stack

The encapsulation of SSL control protocols by the record protocol means that if an active session is renegotiated the control protocols will be transmitted securely. If there were no session before, then the Null cipher suite is used, which means there is no encryption and messages have no integrity digests until the session has been established.

Data Transfer

The SSL Record Protocol, shown in Figure 3, is used to transfer application and SSL Control data between the client and server, possibly fragmenting this data into smaller units, or combining multiple higher level protocol data messages into single units. It may compress, attach digest signatures, and encrypt these units before transmitting them using the underlying reliable transport protocol (Note: currently all major SSL implementations lack support for compression).

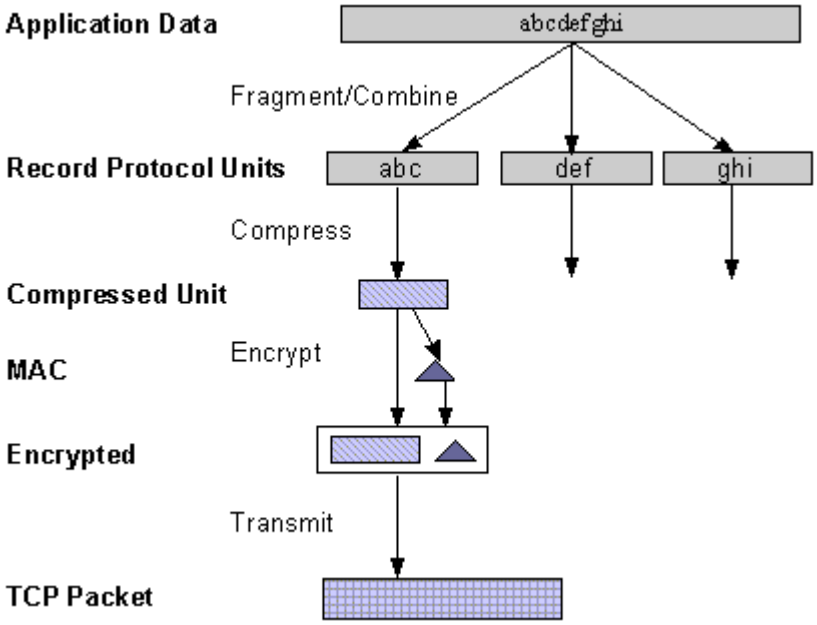


Figure 3: SSL Record Protocol