

SSL e OpenSSL

- Ivan Visconti
- Università degli Studi di Salerno
- visconti@dia.unisa.it



Il protocollo SSL

- **SSL = Secure Socket Layer**
- **Socket = concetto di UNIX per network API**
- **Offre meccanismi di sicurezza e applicazioni che usano il protocollo TCP/IP**
- **E' uno standard per rendere sicuro il protocollo HTTP**
- **Altri protocolli usano SSL (NNTP, POP3, IMAP, ...)**

Caratteristiche di SSL

- **Fornisce l'autenticazione per le applicazioni server e client**
- **Cifra i dati prima di inviarli su un canale pubblico**
- **Garantisce l'integrità dell'informazione**
- **E' stato progettato per essere efficiente**
- **I principali algoritmi crittografici utilizzati vengono negoziati tra le parti coinvolte**

Uso diffuso di SSL

- **Commercio elettronico**
 - >> **Ordinazioni:** le form con cui si ordina un prodotto vengono inviate usando SSL
 - >> **Pagamenti:** quando viene inserito un numero di carta di credito, l'invio dei dati avviene usando SSL
- **Accesso ad informazioni sicure**
 - >> **La consultazione di informazioni accessibili solo da un gruppo chiuso di utenti**
 - >> **L'invio di password o altri dati riservati**

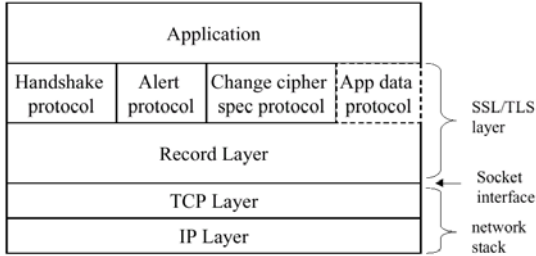
Storia di SSL

- **Sviluppato ed introdotto da Netscape**
- **1994 versione 1: diversi problemi, mai utilizzata**
- **1994 versione 2: implementata in Navigator 1**
- **1996 versione 3: implementata in Navigator 3**
- **1996-1999 TLS (Transport Layer Security) evoluzione di SSL guidata da IETF**

SSL: tecnologie utilizzate

- **Cifratrice simmetrica**
- **Cifratrice asimmetrica**
- **Firme digitali**
- **Certificati digitali (X.509 v.3)**
- **Specifiche chiare e formali**
- **Negoziazione dei parametri**
- **Handshake al momento della connessione**
- **Riutilizzo dei parametri negoziati in precedenza**

SSL tra applicazioni e TCP/IP



SSL e OpenSSL

7

Componenti di SSL

- Alert protocol
 - » Notifica situazioni anomale o segnala eventuali problemi
- Handshake protocol
 - » Permette alle parti di negoziare i diversi algoritmi necessari per la sicurezza delle transazioni
 - » Consente l'eventuale autenticazione tra le parti
- Change Cipher Spec protocol
 - » Impone l'esecuzione di un nuovo handshake per rinegoziare i parametri di sicurezza e l'autenticazione
- Record protocol
 - » Si occupa della compressione, del MAC e della cifratura

SSL e OpenSSL

8

Ciphersuite di SSL

- **ALGORITMO PER LO SCAMBIO DI CHIAVI**
- **ALGORITMO PER L'AUTENTICAZIONE**
- **ALGORITMO PER LA CIFRATURA SIMMETRICA**
- **ALGORITMO PER IL MESSAGE CODE AUTHENTICATION (MAC)**
- **ESEMPI**
 - » EXP- RC4- MD5
 - ⇒ Kx= RSA(512), Au= RSA, Enc= RC4(40), Mac= MD5, exp
 - » DES- CBC3- SHA
 - ⇒ Kx= RSA, Au= RSA, Enc= 3DES(168), Mac= SHA1

SSL e OpenSSL

9

Sessione sicura con SSL

- Una sessione sicura rappresenta una sequenza di valori che possono essere utilizzati con SSL
 - » valori segreti (calcolati durante l'handshake)
 - » ciphersuite (stabilita durante l'handshake)
- Stabilire tutti i parametri ogni volta che c'è una connessione è poco efficiente. Una sessione può sopravvivere quindi tra più connessioni

SSL e OpenSSL

10

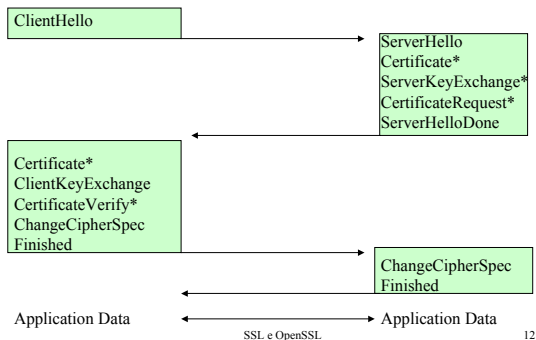
Costo di una nuova sessione

- **CLIENT SIDE**
 - » Generazione di valori random
 - » Controllare la firma digitale del server
 - » Generare dei valori random per la chiave
 - » Cifrare i valori random con la chiave pubblica del server
 - » Calcolare la chiave attraverso degli hash
- **SERVER SIDE**
 - » Generazione di valori random
 - » Decifrare i valori inviati al client
 - » Calcolare la chiave attraverso degli hash

SSL e OpenSSL

11

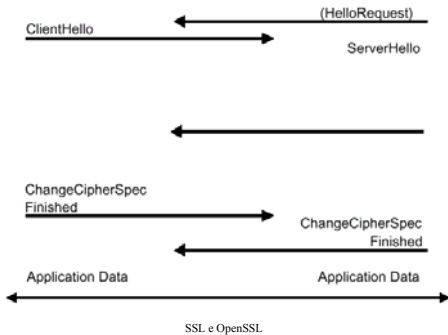
Schema di handshake



SSL e OpenSSL

12

Handshake che riassume una sessione



ClientHello - ServerHello - ServerHelloDone

- **S**ONO I PRIMI MESSAGGI INVIATI PER STABILIRE I PARAMETRI DI UNA SESSIONE
- **P**ERMETTONO DI SCAMBIARE VALORI RANDOM GENERATI DA ENTRAMBE LE PARTI
- **P**ERMETTONO A PARTI DI ACCORDARSI SU UNA CIPHERSUITE
- **C**ONTROLLANO LA NECESSITÀ DI RIRESUMERE UNA SESSIONE INIZIATA IN PRECEDEZZA
- **S**ONO TUTTI **O**BLIGATORI E L'UNICO SENSO DI SERVERHELLODONE È COMUNICARE CHE LA PRIMA PARTE È TERMINATA

SSL e OpenSSL

14

ClientHello

```
struct {
    ProtocolVersion client_version;
    Random random;
    SessionID session_id;
    CipherSuite cipher_suites<0..216-1>;
    CompressionMethod compression_methods<0..28-1>;
} ClientHello;
```

SSL e OpenSSL

15

ServerHello

```
struct {
    ProtocolVersion server_version;
    Random random;
    SessionID session_id;
    CipherSuite cipher_suite;
    CompressionMethod compression_method;
} ServerHello;
```

SSL e OpenSSL

16

Autenticazione

- Questi messaggi consentono alle parti di autenticarsi
- Ognuno contiene una lista di certificati
- Il certificato del server deve essere conforme con l'algoritmo di autenticazione stabilito con la ciphersuite
- Il certificato client deve essere mandato solo se c'è un messaggio CertificateRequest
- L'eventuale certificato inviato dal client deve essere conforme e specifiche indicate nel messaggio CertificateRequest (può vincolare il tipo di certificato e le Certification authority che lo hanno rilasciato)
- Nessuno dei due certificati è obbligatorio ma il certificato server può esserlo in base alla cipher suite stabilita

SSL e OpenSSL

17

KeyExchange

- Il server invia il messaggio ServerKeyExchange se il proprio certificato non è sufficiente per il tipo di autenticazione stabilito nella ciphersuite
- Il messaggio ClientKeyExchange è obbligatorio e con esso le parti hanno le informazioni necessarie per poter calcolare la chiave di cifratura simmetrica da utilizzare dopo l'handshake

SSL e OpenSSL

18

ServerKeyExchange

```
struct {
    select (KeyExchangeAlgorithm) {
        case diffie_hellman:
            ServerDHParams params;
            Signature signed_params;
        case rsa:
            ServerRSAParams params;
            Signature signed_params;
    };
} ServerKeyExchange;
```

ClientKeyExchange

```
struct {
    select (KeyExchangeAlgorithm) {
        case rsa: EncryptedPreMasterSecret;
        case diffie_hellman: ClientDiffieHellmanPublic;
    } exchange_keys;
} ClientKeyExchange;
```

```
struct {
    ProtocolVersion client_version;
    opaque random[48];
} PreMasterSecret;
```

CertificateVerify

- Il messaggio CertificateVerify viene inviato dal client solo se ha inviato il proprio certificato
- Con questo messaggio il client invia una firma digitale dell'hash dei messaggi scambiati fino a quel momento
- Il server non ha un messaggio vero e proprio per dimostrare la propria identità, infatti viene usato il messaggio ServerKeyExchange oppure la decifratura di ClientKeyExchange per dimostrare di avere la chiave privata abbinata alla pubblica evidenziata nel certificato

ChangeCipherSpec - Finished

- Con ChangeCipherSpec ogni parte indica all'altra che sta per usare gli algoritmi e le chiavi appena negoziate
- I messaggi Finished sono obbligatori e sono i primi messaggi che vengono inviati utilizzando gli algoritmi su cui ci si è appena accordati
- La corretta ricezione di tale messaggio dimostra che anche l'altra parte ha calcolato correttamente le chiavi da utilizzare nella cifratura simmetrica

Calcolo delle chiavi

Master_secret =

```
MD5(pre_master_secret +
    SHA('A' + pre_master_secret +
        ClientHello.random + ServerHello.Random)) +
MD5(pre_master_secret +
    SHA('BB' + pre_master_secret +
        ClientHello.random + ServerHello.Random)) +
MD5(pre_master_secret +
    SHA('CCC' + pre_master_secret +
        ClientHello.random + ServerHello.Random))
```

Calcolo delle chiavi

Key_block =

```
MD5(master_secret +
    SHA('A' + master_secret + ClientHello.random +
        ServerHello.Random)) +
MD5(pre_master_secret +
    SHA('BB' + master_secret + ClientHello.random +
        ServerHello.Random)) +
MD5(master_secret +
    SHA('CCC' + master_secret + ClientHello.random +
        ServerHello.Random)) .....
```

Messaggi da firmare in ClientVerify

```
CertificateVerify.signature.md5_hash
    MD5(master_secret + pad_2 +
    MD5(handshake_messages + master_secret + pad_1));
Certificate.signature.sha_hash
    SHA(master_secret + pad_2 +
    SHA(handshake_messages + master_secret + pad_1));
```

pad_1 0x36 ripetuto 48 volte per MD5 o 40 volte per SHA.
pad_2 0x5c ripetuto 48 volte per MD5 o 40 volte per SHA.

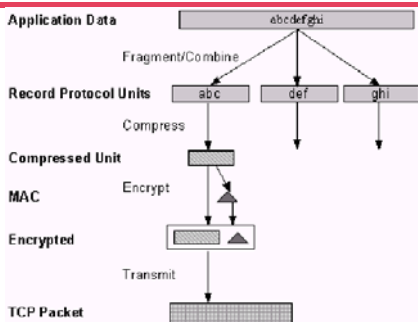
Finished

```
enum { client(0x434C4E54), server(0x53525652) } Sender;
```

```
struct {
    opaque md5_hash[16];
    opaque sha_hash[20];
} Finished;
```

```
md5_hash MD5(master_secret + pad2 + MD5(handshake_messages +
    Sender + master_secret + pad1));
sha_hash SHA(master_secret + pad2 + SHA(handshake_messages +
    Sender + master_secret + pad1));
```

Il compito di SSL



SSL: Analisi

- Connessione anonima
 - » server e client non presentano certificati
 - » scambi di chiavi con Diffie-Hellman
 - » attacco [man in the middle](#)
- Server autenticato
 - » con RSA autenticazione e scambi chiavi combinati
- Server e client autenticati
 - » client deve esplicitamente fornire prova di conoscenza della chiave privata
 - » firma messaggio derivato da master_secret, certificato server e server random

Utilizzo di SSL per il WEB

- Bisogna utilizzare un browser che supporti SSL
- Internet Explorer e Netscape Navigator supportano SSL
- E' possibile installare delle apposite patch per consentire ai browser di utilizzare chiavi a 128 bit per la cifratura simmetrica ed a 1024 bit per la cifratura a asimmetriche
- E' possibile utilizzare un proxy col supporto di SSL

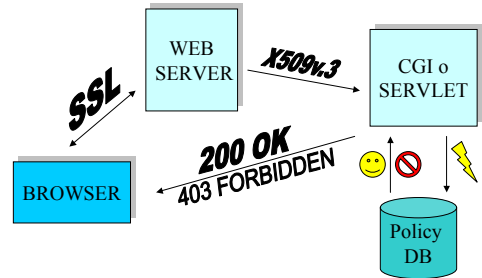
SSL: Access Control

- Autenticazione basata sull'indirizzo dell'host
 - » Soltanto alcuni indirizzi IP hanno l'accesso
 - » Solito problema: Spoofing
 - » Pericolo: Man in the middle
- Autenticazione basata su Cookie e Basic-HTTP
 - » Tutto ciò che passa in rete è cifrato
 - » Viene evitato lo spoofing
 - » Ma i cookie sono in chiaro su disco
 - » Le password sono facilmente comunicate ad altri

SSL: Access Control

- Il server ha un database con i certificati degli utenti qualificati e le politiche di accesso
- Il web server richiede il certificato client durante l'handshake di SSL
- Il client invia il certificato richiesto dal server, di conseguenza, dopo aver stabilito la transazione sicura, potrà ottenere tutti i servizi consentiti dalle politiche di accesso in base all'identità evidenziata dal certificato digitale

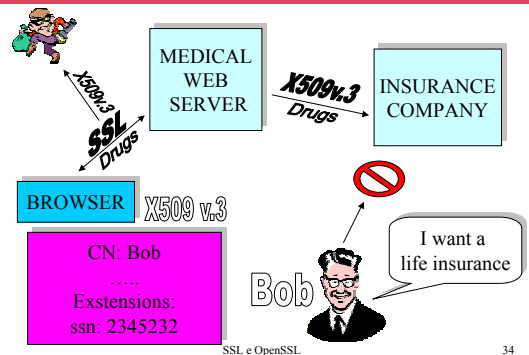
La soluzione basata su servlet o CGI



Quali sono i problemi ?

- Il certificato dell'utente viene inviato in chiaro
- Il server può effettuare il log delle transazioni dell'utente
- Il certificato può contenere informazioni non obbligatorie per la correttezza della transazione
- Il miglior modello prevede una fiducia minimale verso gli altri

SSL trappole per la privacy dell'utente



Disponibilità di SSL

- Implementazione di Netscape
 - » ssref
- Implementazione Open Source
 - » openssl.org
 - » libreria per sviluppare applicazioni basate su SSL
- Supportato da Browser e Web Server
 - » https
- Applicazioni *SSL aware*
 - » telnet

OpenSSL

- OpenSSL è un package open source
- E' sottoposto a manutenzione continua
- E' utilizzata per sviluppare applicazioni molto utilizzate
- Contiene implementazioni di vari algoritmi di crittografia
- Contiene implementazioni di Big Number, formati DER, PEM....
- Implementa il protocollo SSL/TLS
- Ha i comandi per gestire certificati digitali

OpenSSL – Certificati self-signed

- `openssl req -config openssl.cnf -newkey rsa:512 -days 1000 -nodes -keyout cakey.pem -out cacert.pem -x509 -new`
 - » `req` indica la richiesta di un nuovo certificato
 - » `x509` indica che il certificato deve essere self-signed
 - » `config` indica il file con le configurazioni da usare per default
 - » `newkey` specifica il formato della chiave
 - » `days` indica la durata di validità
 - » `nodes` indica che la chiave privata sia salvata in chiaro
 - » `keyout` indica il nome del file con la chiave privata
 - » `out` indica il nome del file col certificato

OpenSSL – richiesta di un certificato

- `openssl req -new -newkey rsa:512 -nodes -keyout Key.pem -out Req.pem -config openssl.cnf`
 - » `new` indica che è una nuova richiesta di certificato
 - » `config` indica il file con le configurazioni da usare per default
 - » `newkey` specifica il formato della chiave
 - » `nodes` indica che la chiave privata sia salvata in chiaro
 - » `keyout` indica il nome del file con la chiave privata
 - » `out` indica il nome del file col certificato

OpenSSL – Rilascio di un certificato

- `openssl ca -policy policy_anything -out cert.pem -config openssl.cnf -infiles req.pem`
- » `config` indica il file con le configurazioni da usare per default
 - » `policy` indica le politiche da utilizzare per il rilascio
 - » `infiles` indica il nome del file con la richiesta
 - » `out` indica il nome del file col certificato
 - » `ca` è l'opzione per la firma di un certificato

OpenSSL – Conversione in p12

- Il formato PKCS12 viene utilizzato per importare certificati e chiavi in un browser
- `openssl pkcs12 -export -chain -CAfile cacert.pem -inkey Key.pem -name Abc -in Cert.pem -out Cert.p12`
- Vengono indicati i file con le informazioni necessarie per ottenere un file in formato PKCS12

SSLClient con OpenSSL

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <netinet.h>
#include <openssl/ssl.h>
#define LEN 1024
#define PORT 3000

main(int argc, char **argv){

    char CAfile[]="CAcert.pem";
    char buff_out[]="PROVA DI INVIO SICURO";
    char buff_in[LEN];
    SSL *ssl;
    SSL_CTX *ctx;
```

SSLClient con OpenSSL

```
int sp, cnt;
struct sockaddr_in servaddr;
if (argc != 2) {
    printf(stderr, "usage: %s <ip address>\n", argv[0]);
    exit(1);
}
if ((sp = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
    perror("opening socket");
    exit(1);
}
bzero(&servaddr, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_port = htons(PORT);
if (inet_pton(AF_INET, argv[1], &servaddr.sin_addr) <= 0) {
    perror("inet_pton error");
    exit(1);
}
```

SSLClient con OpenSSL

```
SSLay_add_ssl_algorithms(); //initialize the supported algorithms
ctx = SSL_CTX_new(SSLV3_client_method()); // create a secure context
SSL_CTX_load_verify_locations(ctx, CAfile, NULL);
ssl = SSL_new(ctx); // create a free and secure connection

SSL_set_fd(ssl, sp); // assign a file descriptor

if (connect(sp, (struct sockaddr *) &servaddr, sizeof(servaddr)) < 0){
    perror("connect error");
    exit(1);
}

SSL_set_verify(ssl, SSL_VERIFY_PEER, NULL);
if (SSL_connect(ssl) < 0){
    printf(stderr, "Error in SSL_Connect\n");
    exit(1);
}

SSL_CTX_free(ctx); // free memory
}
```

SSL e OpenSSL

43

SSLClient con OpenSSL

```
// Do a secure and private connect
SSL_write(ssl, buff_out, strlen(buff_out)); // Do a secure write
SSL_read(ssl, buff_in, LEN); // Do a secure read
printf("Received: %s\n", buff_in);
SSL_shutdown(ssl); // close a secure connection
SSL_free(ssl); // free memory
SSL_CTX_free(ctx); // free memory
}
```

SSL e OpenSSL

44

SSLServer con OpenSSL

```
#include <stdio.h>
#include <stdlib.h>
#include <netinet.h>
#include <openssl/ssl.h>
#define LEN 1024
#define PORT 3000

main(int argc, char **argv){
    char filename[] = "certs/server.pem";
    char secretkey[] = "key/server.pem";
    char buff[LEN];
    SSL *ssl;
    SSL_CTX *ctx;
    int sp, connsp, cnt, lencliaddr;
    struct sockaddr_in servaddr, clientaddr;
}
```

SSL e OpenSSL

45

SSLServer con OpenSSL

```
if ((sp = socket(AF_INET, SOCK_STREAM, 0)) < 0){
    perror("opening socket");
    exit(1);
}

bzero(&servaddr, sizeof(servaddr));
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
servaddr.sin_port = htons(PORT);

if (bind(sp, (struct sockaddr *) &servaddr, sizeof(servaddr)) < 0){
    perror("Error in binding");
    exit(1);
}

listen(sp, 5);
```

SSL e OpenSSL

46

SSLServer con OpenSSL

```
SSLay_add_ssl_algorithms(); // initialize the supported algorithms
ctx = SSL_CTX_new(SSLV3_server_method()); // create a secure context

// Certificate to be used
if (!SSL_CTX_use_certificate_file(ctx, filename, SSL_FILETYPE_PEM)){
    printf(stderr, "Non trovato certificato in %s\n", filename);
    exit(1);
}

// Private key of the certificate
if (!SSL_CTX_use_private_key_file(ctx, secretkey, SSL_FILETYPE_PEM)){
    printf(stderr, "Non trovato chiave in %s\n", filename);
    exit(1);
}

SSL_CTX_free(ctx); // free memory
}
```

SSL e OpenSSL

47

SSLServer con OpenSSL

```
for(;;){
    ssl = SSL_new(ctx); // create a free and secure connection
    connsp = accept(sp, (struct sockaddr *) &clientaddr, (void *) &lencliaddr);
    SSL_set_fd(ssl, connsp); // assign a file descriptor
    // Do a secure accept
    if (SSL_accept(ssl) < 0){
        printf(stderr, "Error in SSL_Accept\n");
        exit(1);
    }

    cnt = SSL_read(ssl, buff, LEN); // Do a secure read
    buff[cnt] = 0;
    SSL_write(ssl, buff, cnt+1);
    SSL_shutdown(ssl); // close a secure connection
    SSL_free(ssl);
}

SSL_CTX_free(ctx); // free memory
}
```

SSL e OpenSSL

48

SSLClient con OpenSSL – Autenticazione client

```
SSL_CTX_load_verify_locations(ctx,CAfile,NULL);
// Certificate to be used
if(!SSL_CTX_use_certificate_file(ctx, filename, SSL_FILETYPE_PEM)){
    fprintf(stderr, "Non trovo certificato in %s\n", filename);
    exit(1);
}

// Private key of the certificate
if(!SSL_CTX_use_private_key_file(ctx, secretkey, SSL_FILETYPE_PEM)){
    fprintf(stderr, "Non trovo chiave in %s\n", filename);
    exit(1);
}
```

SSL e OpenSSL

49

SSLServer con OpenSSL – Autenticazione client

```
SSL_CTX_load_verify_locations(ctx,CAfile,NULL);
SSL_CTX_set_verify(ctx,
SSL_VERIFY_PEER|SSL_VERIFY_FAIL_IF_NO_PEER_CERT, NULL);
...
...
cert=SSL_get_peer_certificate(ssl);
x509_name_oneline(x509_get_issuer_name(cert),buff,LEN);
fprintf(stderr,"Issuer: %s\n",buff);
x509_name_oneline(x509_get_subject_name(cert),buff,LEN);
fprintf(stderr,"Subject: %s\n",buff);
name = strstr(buff,"CN=")+4;
fprintf(stderr,"Common name: %s\n",name);
if(strcmp(name,"client")!=0){
    fprintf(stderr,"Accesso non autorizzato\n");
    exit(1);
}
```

SSL e OpenSSL

50

OpenSSL – Le utility s_server e s_client

- Le utility s_server e s_client vengono distribuite con OpenSSL e sono uno dei principali strumenti di debug utilizzati da chi sviluppa applicazioni client/server sicure
- Possono essere eseguite “indipendentemente” l’una dall’altra e sono configurabili con sequenza di argomenti che consentono di scegliere il tipo di connessione SSL desiderata

SSL e OpenSSL

51

L’utility s_server

- L’utility s_server è parte del package OpenSSL
- E’ un server SSL utile per il debug di applicazioni client col supporto di SSL
- E’ possibile configurare l’esecuzione di questa utility impostando degli argomenti nella riga di comando
- Prevede ad esempio l’uso eventuale di certificati, autenticazione client, selezione di cipher suite, della versione del protocollo

SSL e OpenSSL

52

L’utility s_server – parametri per l’esecuzione

-accept arg	porta TCP/IP del server (default 4433)
-verify arg	richiede l’autenticazione client
-Verify arg	fallisce la connessione se non c’è autenticazione client
-cert arg	indica il file col certificato server (default server.pem)
-key arg	indica il file con la chiave privata (default server.pem)
-dcert arg	eventuale secondo certificato (in generale DSA)
-dkey arg	eventuale seconda chiave (in generale DSA)
-dhparam arg	file con i parametri DH
-nbio	l’esecuzione avviene con socket non bloccante
-debug	vengono visualizzate maggiori informazioni per il debug

SSL e OpenSSL

53

L’utility s_server – parametri per l’esecuzione

-CApath arg	directory con i certificati delle CA
-CAfile arg	file con i certificati delle CA
-nocert	i certificati non vengono utilizzati (Anon-DH)
-cipher arg	uso di particolari cipher suite
-ssl2	uso di SSLv2
-ssl3	uso di SSLv3
-tls1	uso di TLSv1
-no_ssl2	non uso di SSLv2
-no_ssl3	non uso di SSLv3
-no_tls1	non uso TLSv1
-no_dhe	non uso di ephemeral DH
-www	Risposta a GET / con una pagina di prova
-WWW	Risposta a 'GET /<path> HTTP/1.0' con il file ./<path>

SSL e OpenSSL

54

L'utility s_client - parametri per l'esecuzione

-connect host:port	indica il server desiderato (default localhost:4433)
-verify arg	imposta la verifica del certificato del server
-cert arg	indica il certificato da usare
-key arg	indica la chiave da usare
-CApath arg	indica la directory con i certificati delle CA
-CAfile arg	indica il file con i certificati delle CA
-reconnect	interrompe la connessione e la riprende (riesuma)
-showcerts	mostra i certificati ricevuti
-debug	visualizza maggiori informazioni
-nbio	usa socket non bloccanti
-ssl2/ssl3/tls1	imposta un solo protocollo
-no_tls1/-no_ssl3/-no_ssl2	disabilita qualche protocollo
-cipher	specifica le cipher suite

L'utility x509 per la gestione dei certificati

- L'utility x509 di OpenSSL gestisce i certificati digitali
- Permette la conversione tra formati di certificati
- Consente la visualizzazione delle informazioni contenute in un certificato
- Permette di conoscere l'hash di un certificato da utilizzare per referenziarlo come certificato di un Certification Authority in una directory

Uso dell'utility x509

- Ecco le principali opzioni dell'utility:
 - » -in indica il file di input col certificato
 - » -out indica il file di output col certificato
 - » -inform indica il formato di input
 - » -outform indica il formato di output
 - » -text visualizza le informazioni contenute nel certificato
 - » -noout non visualizza il certificato nel suo formato
 - » -hash visualizza l'hash del certificato nel formato necessario per usarlo come una CA in una directory