Creazione di una Certificate Authority

Aniello Castiglione
Gerardo Maiorano
SSG
{anicas,germai}@cert.unisa.it

8 Aprile 1998

Indice

Indice delle Figure

Indice delle Tabelle

1 Introduzione

1.1 La necessità di comunicazioni sicure

Negli ultimi decenni la massificazione delle comunicazioni ha messo in luce il bisogno di comunicazioni sicure; quando si comunica via telefono, fax, email su una linea di trasmissione non protetta, chiunque può intercettare la comunicazione ed avvalersi delle informazioni trasferite. Ciò succede perché ne coloro che comunicano ne chi fornisce il mezzo di comunicazione si preoccupa di proteggere il canale.

1.1.1 Insicurezza nelle reti di comunicazioni

È sempre più difficile stabilire quali sono le informazioni private che possono essere intercettate, tuttavia, non è completamente sbagliato aspettarsi un utilizzo illecito delle informazioni carpite. Nonostante tale scenario possa sembrare poco probabile, in questa area è preferibile prendere delle precauzioni. Consideriamo le principali reti pubbliche di comunicazioni:

Trasmissioni radio e satellitari:

molti sistemi di telefonia analogica cellulare non presentano meccanismi di codifica cifrata tali da evitare intercettazioni da parte di terzi: ultimamente sono venuti alla luce molti casi di chiamate intercettate e registrate.

La maggior parte delle trasmissioni digitali, anche se non sono facili da decodificare, non usano meccanismi di cifratura. Le nuove trasmissioni satellitari digitali utilizzano delle semplici tecniche di cifratura che vengono facilmente eluse.

Telecomunicazioni terrestri:

le reti pubbliche di telefonia terrestre presentano anche loro delle vulnerabilità. In alcune reti (con codifica CCITT7) ci si può inserire nelle centrali numeriche e ridirigere, chiudere, o ascoltare qualsiasi conversazione o trasmissione dati. Questo fenomeno chiamato *Phreaking* preventivato in bilancio dalle compagnie telefoniche, non viene ammesso ufficialmente perché condurrebbe ad una maggiore perdita economica.

Reti Digitali ad alta velocità:

Anche questo tipo di reti non è immune da tali problemi. Ad esempio sulla rete Internet, che è largamente distribuita sul territorio, non dovrebbe essere troppo difficile catturare i pacchetti che transitano per un certo nodo, nonostante sia complicata la ricombinazione dei pacchetti nel loro ordine corretto per riottenere l'intera informazione.

1.1.2 Tipi di Informazione da proteggere

Analizziamo adesso alcuni tipi di informazione che possono richiedere una rete sicura utilizzando la cifratura dei dati.

Transazioni finanziarie

L'intercettazione di transazioni finanziarie non cifrate si pone come minaccia diretta verso la società. Quindi la forza che sta dietro questo attacco non è leggere i dati ma nel cambiarli. Ad esempio, se Assuntina nel trasferire della moneta elettronica a Biagino non utilizza un sistema crittografico forte, Onofrio potrebbe intercettare il messaggio e cambiare il destinatario ed eventualmente l'ammontare.

Siccome la maggior parte dei trasferimenti interbancari sono cifrati (anche se con sistemi di cifraturi deboli, DES a 56 bit), modificare i dati e farli passare per un trasferimento valido è molto difficile ma non impossibile.

Informazioni di carattere commerciali

La maggior parte delle imprese commerciali utilizza i computer e le reti per aumentare la produttività. Di conseguenza, le strategie commerciali future, i dettagli della produzione, le scelte di marketing, i segreti industriali, etc, rappresentano dei potenziali obiettivi che la concorrenza potrebbe provare a carpire. Ad esempio, sapere in anticipo che una certa azienda sta per acquisire un'altra azienda quotata in borsa, darebbe la possibilità di acquistare le azioni di questa azienda prima che esse si rivalutino a causa dell'operazione finanziaria.

Diritto alla privacy

Giustificare l'uso della cifratura come diritto di fatto degli individui è più una questione politica che accademica. Le persone che credone che la libertà di espressione, la libertà di comunicare con chiunque ci pare ed il diritto alla privacy, siano principi base per vivere in una democrazia moderna, vede nella cifratura di tutte le reti di comunicazione pubblica un modo di proteggere tale democrazia.

La potenza della crittografia, però, può essere sfruttata anche a fini non legali: ad esempio un trafficante di droga potrebbe inviare un messaggio cifrato al proprio fornitore in cui ordina una partita di droga; analogamente, gruppi di pedofili potrebbero scambiarsi informazioni circa la loro attività utilizzando la crittografia per nascondere la loro identità. In entrambi i casi, poter intercettare tali messaggi sarebbe un vantaggio per i tutori dell'Ordine Pubblico.

Questi ed altri innumerevoli esempi vengono addotti come motivazione delle restrizioni imposte dai Governi sull'utilizzo della Crittografia. D'altro canto tali restrizioni favoriscono anche i Servizi Segreti di un Paese, permettendo loro di controllare le comunicazioni in modo da proteggere la sicurezza nazionale.

2 Schemi per la cifratura dei dati

Gli schemi per la cifratura dei dati si dividono in due categorie: convenzionali e a chiave pubblica.

- La cifratura convenzionale, conosciuta anche come crittografia a chiave simmetrica, richiede che il mittente ed il destinatario condividano la stessa chiave, cioè un pezzo di informazione segreta che può essere usata per cifrare o decifrare un messaggio. Se questa chiave è segreta nessun altro tranne il mittente ed il destinatario possono leggere il messaggio. Tuttavia il compito di scambiarsi una chiave prima di comunicare rappresenta un problema.

- Gli schemi di cifratura a chiave pubblica prevedono due diverse chiavi: una per la cifratura ed una per la decifratura dei messaggi. Ciò rende possibile ricevere messaggi sicuri semplicemente pubblicando una chiave (la chiave pubblica) e tenendo segreta l'altra (la chiave privata). Chiunque può cifrare un messaggio utilizzando la chiave pubblica, ma solo il possessore della chiave privata associata potrà leggere il messaggio. In questo modo Assuntina può inviare dei messaggi privati a Biagino cifrandoli con la chiave pubblica di Biagino, e viceversa Biagino può inviare dei messaggi privati ad Assuntina cifrandoli con la chiave pubblica di Assuntina.

2.1 Crittografia simmetrica

La maggior parte delle ricerche svolte in questo secolo hanno riguardato la crittografia simmetrica. Un cifrario simmetrico usa una sequenza segreta di caratteri per codificare un testo in chiaro in un unico testo cifrato, cioè una chiave differente usata per codificare lo stesso testo produce un testo cifrato completamene diverso. Il testo in chiaro è riottenuto riapplicando lo stesso algoritmo con la stessa chiave sul testo cifrato.

Gli algoritmi simmetrici si dividono in due categorie: *block cipher* e *stream cipher*.

- I block cipher prendono in input un blocco di lunghezza fissa del messaggio e generano in output un blocco cifrato della stessa lunghezza. Se la lunghezza totale del testo in chiaro non è un multiplo della dimensione del blocco, allora si aggiugono all'ultima parte del testo in chiaro alcuni bit fino ad ottenere la dimensione del blocco (padding).

- I stream cipher operano su piccole unità del testo in chiaro, tipicamente bit. La cifratura di un blocco di testo in chiaro con un block cipher produce sempre lo stesso blocco cifrato se si utilizza la stessa chiave; con uno stream cipher,invece, la trasformazione delle unità elementari di testo in chiaro può variare a seconda del punto in cui il processo di cifratura le incontra.

2.2 Crittografia asimmetrica

La crittografia a chiave pubblica fu inventata da Whitfield Diffie e Martin Hellman nel 1976, mentre il sistema di crittografia a chiave pubblica più conosciuto, chiamato RSA, fu sviluppato nel 1978 da Ron Rivest, Adi Shamir e Leonard Adleman.

Come abbiamo già sottolineato, i sistemi di crittografia a chiave pubblica differiscono da quelli simmetrici in quanto usano una chiave privata per decifrare e una chiave pubblica per cifrare, anziché usare un'unica chiave per entrambe le operazioni.

Sebbene le chiavi pubbliche e private siano correlate matematicamente, la chiave privata non può essere facilmente ricavata dalla chiave pubblica e per questo motivo rendere nota la chiave pubblica non costituisce una perdita di sicurezza. La sicurezza del sistema RSA si basa sulla difficoltà computazionale di fattorizzare interi molto grandi ed è pertanto molto iportante scegliere con cura, la dimensione degli interi scelti come base del sitema di cifratura. Attualmente si consiglia di usare chiavi a 1024 bit (368 cifre). Le funzioni di cifratura e decifratura di uno schema a chiave pubblica come l'RSA possonoessere usate per ottenere firme digitali.

Message digest. Prima di analizzare le firme digitali è utile introdurre un altro strumento detto *message digest* o *one-way hash*. Un message digest è una sorta di riassunto di un messaggio ed è usato per creare delle piccole rappresentazioni a lunghezza fissa di un messaggio grande di lunghezza variabile. Gli algoritmi di *digest* sono progettati per produrre output differenti su input differenti, per rendere difficile risalire al messaggio originale noto il digest, ed è anche impossibile trovare due messaggi differenti che danno luogo allo stesso digest, eliminando così la possibilità di sostituire un messaggio con un altro mantenendo lo stesso digest.

Quando Biagino riceve un messaggio da Assuntina, ha bisogno di assicurarsi che il messaggio realmente proviene da Assuntina. A tale scopo Assuntina deve accludere al messaggio la sua firma digitale. La firma digitale è creata cifrando un digest del messaggio da inviare (insieme ad altre informazioni) con la chiave privata del mittente.

Chiunque può verificare che tale firma appartiene al mittente utilizzando la chiave pubblica del mittente (nota a tutti), ricavando il digest del messaggio. Il ricevente può assicurarsi dell'integrità del messaggio ricalcolandone il digest e confrontarlo con quello contenuto nella firma digitale, poiché nessuno può cambiare il digest e rifirmarlo. Per proteggersi dalla intercettazione e dal riutilizzo seguente del messaggio da parte di un intruso, la firma contiene un numero di serie. La firma digitale evita che Assuntina possa disconoscere l'invio di un messaggio poiché solo lei può averlo firmato (non-repudiation).

3 Certificati Digitali

Una volta reso sicuro il canale di communicazione, il prossimo compito che ci aspetta è l'autenticazione. L'autenticazione consiste nell'assicurarsi che la persona con cui si sta parlando è effettivamente la persona con cui si vuol parlare.

Il seguente è un tipico esempio in cui la necessità di autenticazione mutua tra le parti riselta evidente. Supponiamo che Assuntina e Biagino intendano comunicare usando un sistema a chiave pubblica e pertanto Biagino necessita della chiave pubblica di Assuntina. Onofrio intercetta la richiesta di Biagino di conoscere la chiave pubblica di Assuntina e invia la sua chiave pubblica spacciandola per quella di Assuntina. Biagino usa la chiave pubblica di Onofrio per cifrare la sua conversazione, pensando che si tratti della chiave pubblica di Assuntina. Onofrio può ora intercettare tutti i messsaggi di Biagino diretti ad Assuntina, decifrarli, ricifrarli usando la chiave pubblica reale di Assuntina e spedirli ad Assuntina. Una intera comunicazione può avvenire in questo modo senza che né Assuntina né Biagino si accorga di nulla.

Per permettere l'identificazione sicura sulla rete è stato introdotto il concetto di *certificato digitale*. Un certificato digitale svolge nel mondo della rete un compito analogo a quello di un documento d'identità. Un passaporto, lega un nome ad una fotografia mediante il timbro di una autorità riconosciuta. Similmente, un certificato lega un nome ad una chiave pubblica mediante la firma digitale di una autorità riconosciuta, in questo caso denominata Certificate Authority (CA). Un certificato, tipicamente, consiste nel nome e la chiave pubblica della soggetto, e il nome, la chiave pubblica e la firma digitale della Certificate Autority. La firma del certificato è calcolata dalla CA sul nome e sulla chiave pubblica del soggetto (per motivi di efficienza, si firmano solo un digest di questi dati) usando la chiave privata della CA. Il certificato può essere verificato in qualunque momento da chiunque decifrando il blocco della firma con la chiave pubblica della Certificate Authority e confrontando il risultato con il digest ricalcolato da dati del certificato. Ovviamente, affinché questa verifica possa essere effettuata, è necessario che la chiave pubblica della CA sia accessibile a tutti.

In realtà tutti i certificati trasferiscono il problema della fidatezza dalla persona ad una CA. Così il problema diventa: come si fa a capire che la chiave pubblica di una CA appartiene proprio a lei? La risposta è ovvia: tramite un certificato per la CA.

Onofrio, nel caso in cui Assuntina e Biagino utilizzino dei certificati per autenticarsi a vicenda, non può più identificarsi come uno dei due. Infatti, per fare ciò, dovrebbe essere capace di costruire un certificato con il nome del soggetto che vuole impersonare e la propria chiave pubblica. Ma ciò è impossibile in quando per tale certificato non potrebbe produrre la firma della CA. Sarebbe come avere un passaporto con la propria foto e un'altro nome, ma senza il timbro dell'autorità ricosciuta. La crittografia a chiave pubblica garantisce che il "timbro" sul certificato non si possa falsificare.

3.1 Contenuto di un certificato X509.v3

Come già esposto in precedenza, un certificato associa una chiave pubblica con l'identità reale di un individuo, server o altra entità. In questa sezione discutiamo i campi presenti in un Certificato Digitale come specificato dallo standard X509.v3. Questa identità è detta Soggetto (Subject). Come mostrato nella tabella 1, le informazioni riguardanti il Soggetto consistono nelle informazioni di identificazione (Distinguished Name), e nella chiave pubblica da esso generata. Il certificato contiene anche l'identificazione e la firma della Certificate Authority che ha emesso (Issuer) il certificato, e il periodo di validità del certificato stesso; inoltre, esso può anche contenere delle informazioni addizionali di carattere amministrativo e delle estensioni che facilitano l'uso del certificato da parte delle applicazioni software.

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	

Tabella 1: Informazioni contenute nel certificato

I Distinguished Name sono definiti dallo standard X.509, che definisce i campi, i nomi dei campi, e le abbreviazioni usate per riferirsi ai campi (vedi tabella 2).

Field	**Abbr.**	**Description**
Common Name	CN	Name being certified
Organization or Company	O	Name is associated with this organisation
Organisation Unit	OU	Name is associated with this organisation unit
Locality	L	Name is located in this City
StateorProvince	SP	Name is located in this State or Province
Country	C	Name is located in this country (ISO code)

Tabella 2: Informazioni del Distinguished Name

Una Certificate Authority può definire una politica che illustri quali nomi di campi sono opzionali e quali sono richiesti, che tipo di contenuto deve avere un certo campo e cosi via. Per esempio, un browser Netscape richiede che il Common Name del certificato, presentato da un server durante una connessione, abbia un valore che faccia match con l'espressione regolare del nome del dominio del server (ad esempio *.unisa.it).

Il formato binario di un certificato è definito usando la notazione ASN.1. Questa notazione definisce come specificare il contenuto, e le regole di codifica che definiscono come queste informazioni sono tradotte nella forma binaria. La codifica binaria del certificato è definita usando le Distinguished Encoding Rules (DER), che sono basate sulle più generali Basic Enconding Rules (BER). Per quelle trasmissioni che non possono manipolare dati binari, la forma binaria può essere tradotta in una forma ASCII equivalente usando la codifica base64. Questa codifica, quando è inserita tra le due linee seguenti:

```
-----BEGIN CERTIFICATE-----
-----END CERTIFICATE-----
```

è detta codifica PEM (Privacy Enhanced Mail).

3.2 Compiti di una CA

La CA rilascia un certificato in presenza di una richiesta di certificato accompagnata da credenziali verificabili che provano l'identita del possessore della chiave privata. Le credenziali che si devono presentare ad una CA per ottenere un certificato, variano a seconda della politica adottata dalla CA.

Una CA può anche emettere un certificato per un'altra CA. Si può così creare una struttura gerarchica di CA partendo da una *root* CA di carattere internazionale fino ad arrivare a CA locali.

Quando esamina un certificato, Assuntina può aver bisogno di esaminare il certificato del garante, per ogni CA della gerarchia, fino ad arrivare ad un certificato di cui si fida. In questo modo avendo una catena di certificati ci si può fidare del certificato che sta all'inizio della catena.

La root CA non ha un garante che firma il suo certificato, e quindi solo in questo caso il certificato è autofirmato: il garante (Issuer) e il soggetto (Subject) del certificato sono la stessa entità.

Ovviamente bisogna fare molta attenzione nel fidarsi di certificati autofirmati non appartenenti a CA note. Una grossa diffusione delle chiavi pubbliche delle root CA riduce il rischio nell'affidarsi a queste chiavi. I browser sono preconfigurati per riconoscere certificati firmati da CA universalmente note.

I servizi principali che una CA deve offrire sono:

1. la verifica delle richieste di certificato

2. l'elaborazione delle richieste di certificato

3. la emissione e la gestione dei certificati.

Quindi fondare una CA è una responsabilità che richiede una infrastruttura amministrativa, tecnica e di gestione molto solida. La CA non solo emette i certificati ma li deve anche gestire, cioè, determinare il periodo di validità del certificato, esaminare i rinnovi dei certificati gia emessi e tenere una lista dei certificati che sono stati emessi ma che non sono più validi. Questa lista è detta lista dei certificati revocati (CRL). Se Assuntina possiede un certificato in qualità di impiegata di una certa azienda, se per caso essa lascia l'azienda allora il suo certificato non deve più valere e deve essere inserito nella CRL. Quindi quando si esamina un certificato non si deve controllare solo se esso è valido ma anche se è inserito nella CRL della CA che lo ha emesso.

4 Protocolli di comunicazioni sicure

4.1 Introduzione

Le implementazioni di sistemi di crittografia a chiave pubblica sono molto più lente di quelle di stistemi di crittografia simmetrici. Questo fatto ha spinto i progettisti di protocolli di comunicazione sicura ad usare una combinazione di entrambi i sistemi per ottenere sicurezza e velocità. L'idea consiste nell' utilizzare il sistema a chiave pubblica per lo scambio delle chiavi: tali chiavi sono chiamate *chiavi di sessione.* e sono generate in modo casuale; in seguito vengono usate per cifrare i messaggi con un cifrario simmetrico.

Le chiavi generate in modo casuale giocano un ruolo molto importante nella sicurezza di un qualsiasi sistema di cifratura. Se l'algoritmo progettato per generare queste chiavi non lavora bene, cioè si può in qualche modo prevedere la sequenza delle chiavi generate, allora la sicurezza del cifrario è totalmente compromessa. Quindi, la bontà del sistema di generazione delle chiavi è un parametro da tenere in forte considerazione quando si esamina un protocollo di comunicazione sicura. Un altro parametro importante è la lunghezza della chiave usata dal cifrario simmetrico.

I protocolli che hanno avuto più successo negli ultimi anni sono tra gli altri PGP e SSL.

4.2 PGP (Pretty Good Privacy)

PGP è stato progettato e inizialmente implementato da Phil Zimmermann e ha subito molte revisioni dal 91 ad oggi. PGP utilizza IDEA come cifrario per codificare i messaggi, RSA per la firma di questi ultimi e per lo scambio delle chiavi simmetriche, MD5 per il digest. Le chiavi pubbliche e private generate con RSA possono essere di 512, 768, 1024 bit e nelle ultime implemetazioni anche lunghezze superiori.

4.3 SSL (Secure Socket Layer)

SSL è un protocollo di comunicazione sicura che fornisce comunicazioni private su Internet. Il protocollo permette ad applicazioni client/server di comunicare in un modo che è progettato per prevenire intercettazioni, modifiche, e falsificazioni dei messaggi.

Cenni storici

SSL è stato originariamente progettato ed implementato da Netscape Communication Corporation. Netscape è un'azienda che ha avuto origine in ambito accademico: infatti nel 1993 Marc Andreessen, uno studente 22-enne della University of Illinois, progettò e scrisse Mosaic, il primo browser grafico per il Web. Nell'aprile del 1994 Andreessen e Jim Clark, professore alla Stanford University, fondarono la Netscape ed implementarono un nuovo browser: il Netscape Navigator. Tale browser fu distribuito gratutitamente per uso personale e quindi si diffuse molto velocemente tra gli utenti della Rete. La prima affermazione commerciale di Netscape fu la quotazione in borsa nell'agosto 1995: la società venne valutata per 565 milioni di dollari e ogni azione costava 58.25 dollari. Sin dalla sua fondazione Netscape ha attirato l'attenazione del mondo accademico: in particolare la collaborazione con il noto crittografico Taher ElGamal famoso per aver sviluppato il sistema di crittografia a chiave pubblica che porta il suo nome.

Nei piani dell'azienda c'é sempre stata l'idea di ricavare profitto dal *Commercio Elettronico su Internet*. Questa opportunità non si poteva realizzare prima a causa delle evidenti insicurezze di Internet; per risolvere ciò, Netscape decise di sviluppare un protocollo di comunicazione sicura che permettesse la trasmissione sicura di informazioni riservate come dati personali, numeri di carte di credito, ed altro. Il loro piano commerciale prevedeva l'inserimento di queste caratteristiche *sicure* nei browser senza alcuna spesa, mentre i server che avevano tali caratteristiche venivano venduti a società che li utilizzavano come se fossero dei registratori di cassa su Internet. Il risultato dei loro sforzi progettuali condusse alla pubblicazione di un *Internet Draft* nel 1994, che prese il nome di SSL 2.0.

Alla fine del 1996 Netscape ha rilasciato la versione 3.0 del protocollo che risolveva alcuni problemi della versione precedente. La validità del progetto è stata riconosciuta da tutti e ha portato la IETF a creare un *Working Group* di nome TLS che si occupa della standardizzazione del protocollo.

4.4 Caratteristiche Tecniche di SSL

SSL può essere stratificato al di sopra di un qualsiasi protocollo di trasporto affidabile, non è dipendete da TCP/IP, e può funzionare al di sotto di un qualsiasi protocollo di applicazione come HTTP, FTP, TELNET.

SSL usa lo schema di *certificazione* del ITU (ex CCITT) chiamato X.509 per l'*autenticazione*, RSA, Diffie-Hellman e Fortezza per lo *scambio chiavi*, i *cifrari simmetrici* IDEA, RC2, RC4, DES, 3DES e FORTEZZA e le *funzioni hash* MD5 e SHA.

È importante che ogni nuovo protocollo di comunicazione sia conforme allo standard se vuole facilmente rimpiazzare o prendere parte ad una struttura di protocollo preesistente. Il modello di riferiento ISO per l'Interconnessione di Sistemi Aperti (ISO/OSI) noto anche come modello a sette strati (7-Layer Model), è il modello astratto più popolare.

Seguendo lo standard ISO, l'Application Layer oltre alle informazioni sul trasferimento, dovrebbe avere tra i servizi che fornisce anche un meccanismo per l'autenticazione delle parti che prendono parte alla comunicazione; mentre il Presentation Layer dovrebbe avere la funzionalità di cifratura e decifratura della comunicazione con una chiave nota solo ai partecipanti alla comunicazione.

Ed è proprio qui che si colloca il lavoro di SSL che viene spezzato in due parti proprio come suggerito dal protocollo ISO/OSI: SSL Handshake Protocol (SSLHP) e SSL Record Protocol (SSLRP).

SSLHP sceglie quale cifrario simmetrico debba cifrare il canale di comunicazione e realizza (se richiesta) l'autenticazione del server e del client.

SSLRP ingloba i dati in record e realizza la cifratura dei record inviati e la decifratura dei record ricevuti. SSLHP e l'Application Layer si stratificano sopra SSLRP.

Così, come si può vedere dalla figura, i due strati ISO sono affiancati da quelli di SSL, in modo che SSLHP sta al livello dell'Application Layer e SSLRP sta al livello del Presentation Layer.

Siccome molti protocolli non implementano funzioni per la sicurezza, SSL si comporta come un'aggiunta a tali protocolli e non come un rimpiazzamento. In più si può notare che l'uso di SSL non preclude l'utilizzo di protocolli sicuri che operano a livello più alto: per esempio SHTTP (Secure HyperText Transfer Protocol), che è un protocollo sicuro per i documenti e i dati, indicato in modo specifico per le transazioni finanziarie, può essere stratificato su SSL.

Quando un client ed un server SSL comunicano per la prima volta, devono stabilire la versione del protocollo, selezionare gli algoritmi crittografici da utilizzare, eventualmente autenticarsi tra loro, e utilizzare le tecniche di crittografia a chiave pubblica per generare le informazioni segrete da condividere.

Questi processi sono eseguiti nella fase di *Handshake* dal SSLHP, che si comporta nel seguente modo: il client invia il messaggio *client hello* al server che dovrebbe rispondere con un messaggio *server hello* se ciò non avviene la connessione cade. I messaggi client hello e server hello sono usati per stabilire le caratteristiche di sicurezza possedute dal client e dal server (algorimtmi e metodi di scambio delle chiavi preferiti).

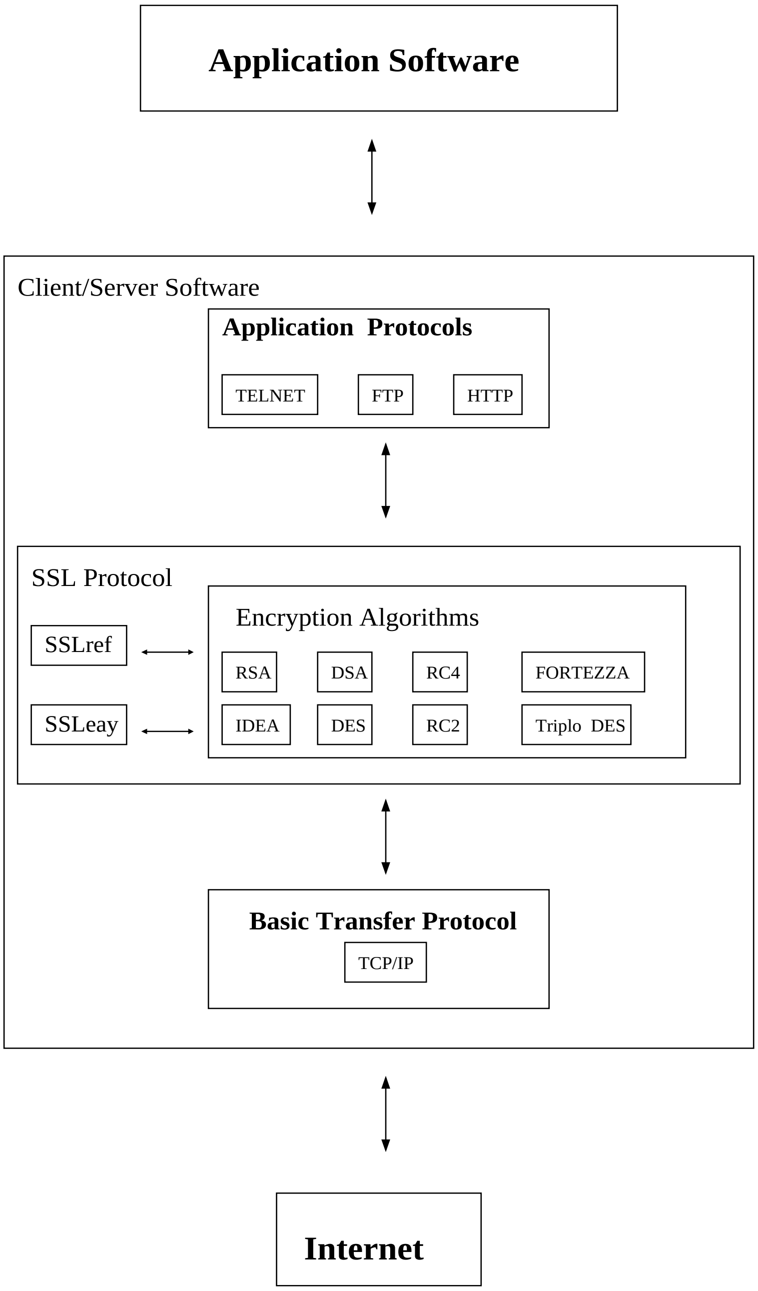

Figura 1: Diagramma delle relazioni tra SSL, le Applicazioni e la Rete

I messaggi client hello e server hello stabiliscono i seguenti attributi: versione del protocollo, identificativo della sessione, algoritmi di crittografia preferiti (*cipher suite*), metodo di compressione dei dati ed in più vengono generati e scambiati due numeri casuali.

Dopo i messaggi di hello, il server se deve essere autenticato invia il suo certificato altrimenti viene inviato un messaggio *server key exchange* per lo scambio delle chiavi di sessione. Ciò avviene quando il server non possiede un certificato od il suo certificato è valido solo per la firma. Se il

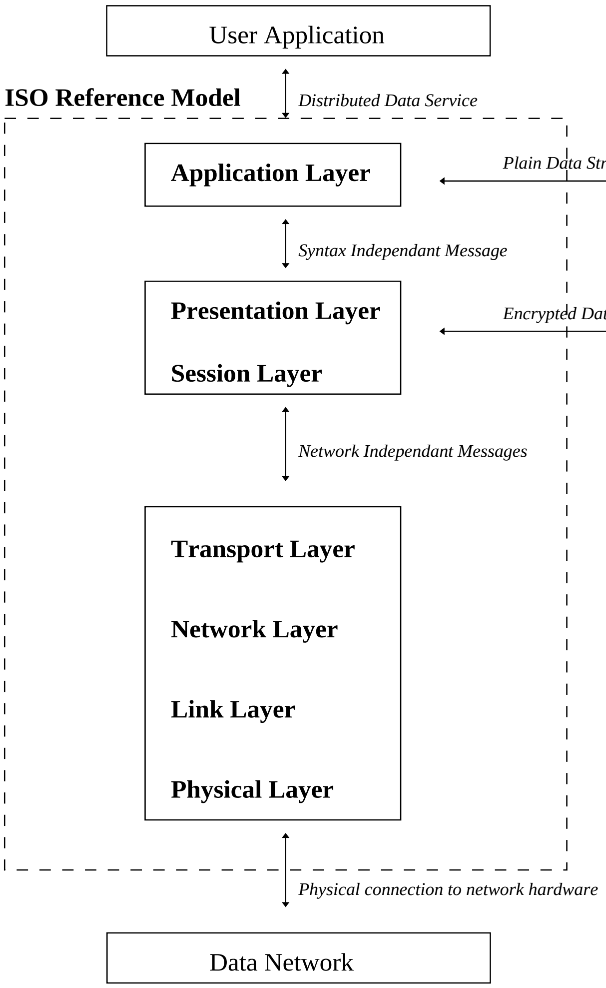

Figura 2: Diagramma delle relazioni tra SSL, e il modello ISO

server è autenticato e la cipher suite scelta lo permette, esso può chiedere al client di autenticarsi presentando un proprio certificato.

Se tutto va bene, il server invia il messaggio *server hello done* che segnala la fine della fase di hello del processo di handshake e si mette in attesa della risposta del client.

Se il server ha richesto al client il certificato tramite il messaggio *certificate request* il client deve inviare in risposta o il messaggio *certificate* contenente il suo certificato oppure il messaggio *no certificate* che è un messaggio di allarme. Il client invia il messaggio *client key exchange* il cui contenuto dipende dall'algoritmo di crittografia a chiave pubblica selezionato nella fase di hello. Se il client possiede un certificato valido anche per la firma, invia un messaggio firmato *certificate verify* che richiede la verifica esplicita del certificato presentato.

A questo punto, il client invia il messaggio *change cipher spec* segnalando che dopo questo messaggio tutti i dati inviati rispetteranno le regole stabilite nelle fasi precendenti riguardo alle chiavi, agli algoritmi, alle funzioni hash e così via. Subito dopo il client invia il messaggio *finished* secondo le convezioni stabilite.

In risposta il server invia anche esso il messaggio *change cipher spec* e il messaggio *finished* con le nuove regole. Ora la fase di handshake è completa e si può iniziare a scambiare i dati. (vedi figura 3)

Ovviamente, la prima volta tutti i messaggi passano in chiaro fino al messaggio *finished*. Se invece il client ed il server decidono di ripristinare una sessione precedente o duplicare una sessione già in atto (invece di rinegoziare dei nuovi parametri), allora i messaggi scambiati sono diversi.(vedi

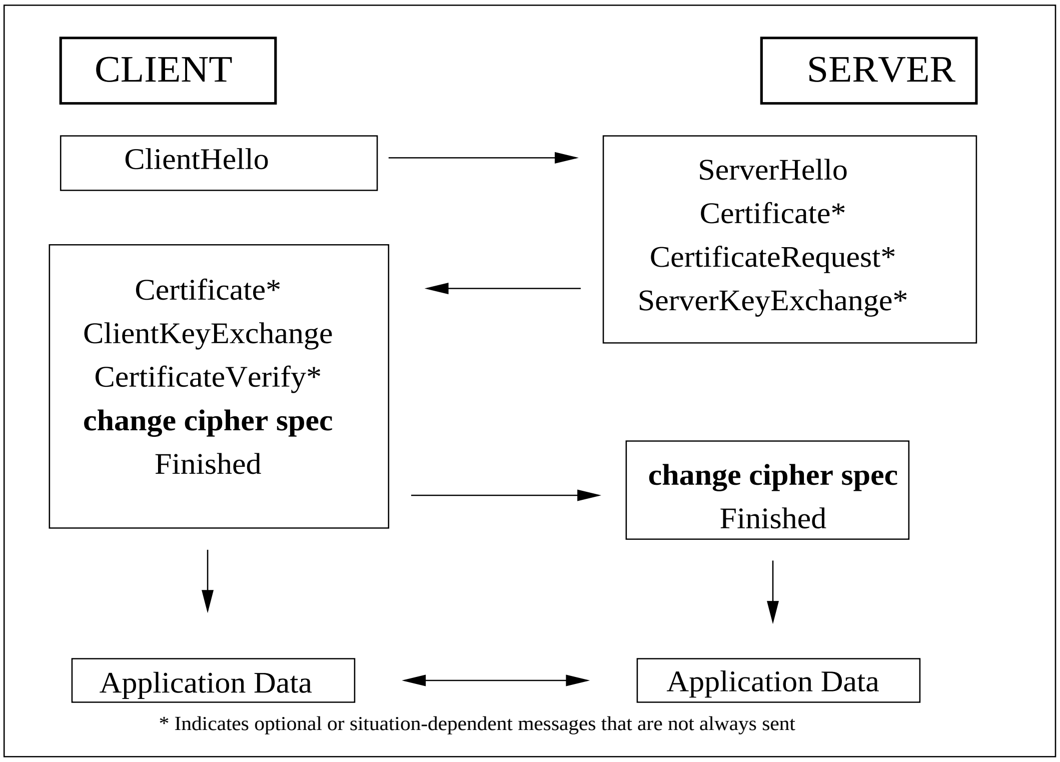

Figura 3: Diagramma di flusso dello handshake completo

figura 4) Il client invia il messaggio di hello con l'identificativo della sessione che si vuole ristabilire. Il server verifica se aveva iniziato una tale sessione, e se è così e se vuole ristabilire la connessione con i parametri negoziati in precendenza, invia un messaggio di *server hello* contenente lo stesso identificativo di sessione. A questo punto, sia il client che il server devono inviare il messaggio *change cipher spec* seguito dal messaggio *finished*. Una volta ristabilita la connessione i dati vengono scambiati secondo le regole prestabilite in precedenza. Se il server non aveva iniziato una sessione con quell'identificativo ovvero non ha più i parametri nella sua cache di sessione allora si esegue un handshake completo.

I dati vengono trasferiti secondo il SSLRP. Un record SSL consiste di due parti, header e dati. L'intestazione (header) può essere lunga due o tre byte a seconda dei casi. La parte dati consiste di un Message Authentication Code (MAC), dei dati da inviare più dei byte prefissati (*padding data*) per raggiungere la lunghezza del record. I padding data sono richiesti solo quando il cifrario utilizzato è un *block cipher*.

Nella fase di handshake il client genera un sequenza di 48 byte detta *master secret* che spedisce la server utilizzando la chiave pubblica del server riportata nel certificato esibito dal server. Questi 48 byte vengono divisi in quattro parti dette *server write MAC secret*, *client write MAC secret*, *server write key*, *client write key*.

Le due *write key* vengono utilizzate per cifrare i due flussi di comunicazione (dal client al server e viceversa) mentre le *MAC secret* vengono utilizzate quando si fa il MAC del record. Il MAC è un hash (o un message digest) di varie componenti: la MAC secret di chi invia il dato, il dato stesso, eventuali padding data, e un numero di sequenza unico per ogni partecipante alla comunicazione: tale numero viene incrementato dopo che ciascun messaggio è spedito evitando possibili attacchi di *replay*.

Per evidenziare meglio il funzionamento di SSL in modo da capire perché esso garantisce l'autenticazione e la privatezza dei dati, esaminiamo un tracciato di una transazione SSL. In particolare si esamina la fase di Handshake, analizzando i messaggi diagnostici forniti delle due applicazioni

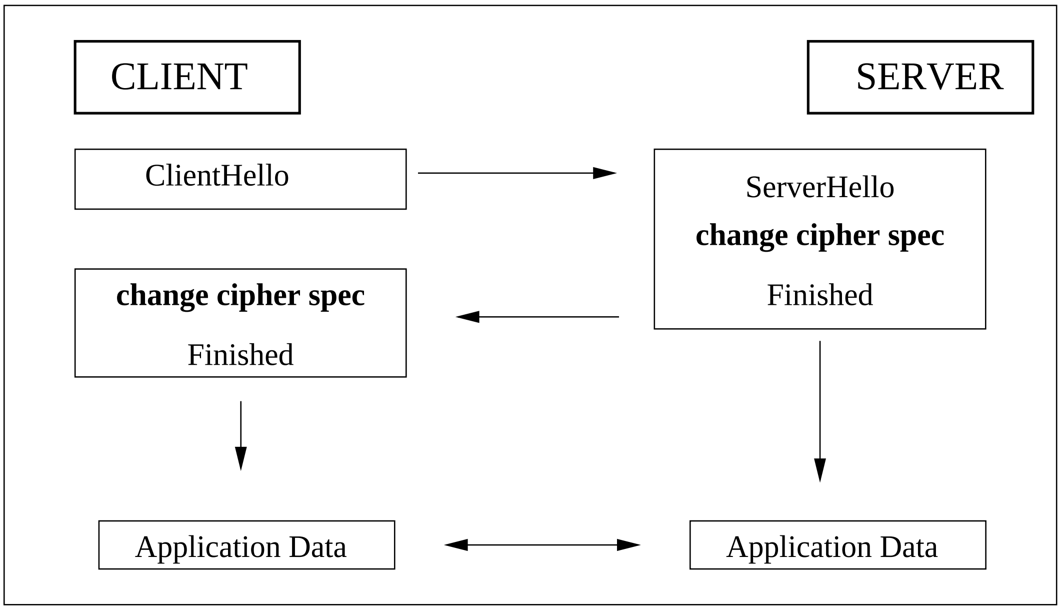

Figura 4: Diagramma di flusso dello hanshake rinegoziato

s_server e **s_client** di SSLeay. I tracciati sono mostrati nelle figure 5,6.

Il comando *s_server* permette di utilizzare un server HTTP minimale che supporta il protocollo SSL. Il parametro *-Verify 3* indica che il client deve fornire necessariamente un certificato e che la verifica del certificato verrà effettuata per non piu' di 3 livelli; *-cert* indica il file che contiene il certificato del server da fornire ai client che si collegano; *-key* indica la chiave privata del server associata alla chiave pubblica specificata dal parametro *-cert*; *-CAfile* contiene i certificati delle CA riconosciute dal nostro server di test; *-state* indica di mostrare l'output di diagnostica, *-ssl3* dice di usare la versione 3 del protocollo SSL.

Il server rimane in attesa di eventuali collegamenti [ACCEPT] (righe 2-4). Non appena arriva un messaggio *Client Hello* (riga 6), il server risponde con il messaggio *Server Hello* e invia il proprio certificato al client (righe 7-8). Siccome si è richiesta l'autenticazione del client il server invia la richiesta di *Certificate Request* e comunica di aver terminato la propria fase (righe 9-10). Il server ricevuto il certificato client (righe 12-16) ne verifica la validità controllando che la firma dello *Issuer* sul certificato sia quella fatta da una CA che egli ricosce; ciò permette l'autenticazione del client. Il server legge il messaggio del client *Client Key Exchange* contenente tra l'altro le chiavi di sessione (riga 17) ed il messaggio di avvenuta verifica del certificato server da parte del client (riga 18) e riceve il messaggio *Finished* sempre dal client (riga 19). A questo punto il server manda i messaggi *Change Cipher Spec* e *finished* (righe 20-21): da questo punto in poi tutta la comunicazione (compreso il messaggio *finished*) che intercorre verrà cifrata utilizzando i protocolli di cifratura stabiliti in precedenza durante i messaggi di hello e di key exchange. Il server evade la richiesta del client che richiedeva **GET / HTTP/1.0** e ritorna in ACCEPT (righe 23,25).

Il comando *s_client* è un client HTTP minimale che supporta SSL. I parametri utilizzati sono analoghi a quelli di *s_server*. Il client inizia la connessione inviando (riga 5) il messaggio di *Client Hello* e leggendo il messaggio di risposta del server *Server Hello* (riga 6), il certificato del server, e la richiesta di fornire il suo certificato (righe 7-12). Verifica la validità controllando che la firma dello *Issuer* sul certificato sia quella fatta da una CA che egli ricosce; ciò permette l'autenticazione del server. Questa fase di autenticazione del server avviene sempre ed è quella che innesca la possibilità di cifrare la comunicazione. Nel nostro esempio, il client si deve per forza autenticare ed è per questo che invia messaggi indicati nelle righe 14-17, leggendo poi la risposta finale del server alla riga 20 che arriva gi'a cifrata secondo l'algoritmo stabilito nel messagio di *change cipher spec*. La privatezza dei

```
% s_server -Verify 3 -cert apache_cert.pem -key apache_key.pem -CAfile
WW -state -ssl3
verify depth is 3, must return a certificate
Using default temp DH parameters
ACCEPT
SSL_accept:before SSL initalisation
SSL_accept:SSLv3 read client hello A
SSL_accept:SSLv3 write server hello A
SSL_accept:SSLv3 write certificate A
SSL_accept:SSLv3 write certificate request A
SSL_accept:SSLv3 write server done A
SSL_accept:SSLv3 flush data
depth=1 /C=IT/ST=Salerno/L=Baronissi/O=Security Service Group/O=Univ.
matica ed Applicazioni/OU=SSG Certificate Authority/CN=SSG Root CA Certif
a.it
verify return:1
depth=0 /C=IT/ST=Salerno/L=Baronissi/O=Security Service Group/OU=SSG C
rtificato client di prova
verify return:1
SSL_accept:SSLv3 read client certificate A
SSL_accept:SSLv3 read client key exchange A
SSL_accept:SSLv3 read certificate verify A
SSL_accept:SSLv3 read finished A
SSL_accept:SSLv3 write change cipher spec A
SSL_accept:SSLv3 write finished A
SSL_accept:SSLv3 flush data
FILE:index.html
SSL3 alert write:warning:close notify
ACCEPT
SSL3 alert read:warning:close notify
```

Figura 5: Tracciato server *s_server*

dati è assicurata dalla cifratura e dal fatto che le chiavi non sono conosciute a nessun altro tranne che al server ed al client anche se si é osservato tutto ció che è transitato tra i due partecipanti.

Futuri sviluppi

Il protocollo SSL garantisce l'autenticazione ma presenta il problema della *privacy*. Infatti, poiché il certificato client passa in chiaro, chiunque può osservare chi si collega ad un certo server e trarre vantaggio da questa informazione. Inoltre il server stesso sa chi si collega e cosa fa e può trasferire tali informazioni a terzi. Sarebbe allora auspicabile che il certificato client passasse cifrato in modo che solo il server se ne potesse servire per l'autenticazione ed inoltre il meccanismo dovrebbe prevedere la possibilità di presentare il client come un individuo di un certo gruppo ma non indicare precisamente di chi si tratta. Ciò assicurebbe sia la privacy che l'autenticazione.

Implementazioni di SSL

Sebbene si possa scrivere una implementazione di SSL a partire da zero, è molto più facile utilizzare delle librerie preconfezionate che includano la cifratura, il message digest e le routine di gestione dei certificati. Inoltre bisogna fare attenzione al fatto che gli algoritmi di cifratura a chiave pubblica e altri strumenti crittografici sono soggetti a brevetti e restrizioni governative, soprattutto negli Stati Uniti.

I pacchetti più conosciuti per la crittografia a chiave pubblica disponibili sono:

RSARef: è una implementazione non più supportata della RSA. Si può usare liberamente per applicazioni non commerciali e i diritti commerciali sono detenuti dalla Consensus Development

Corp.

BSAFE3.0: è una implementazione commerciale di RSARef della RSA.

Queste due implementazioni di crittografia a chiave pubblica includono un kit completo di algoritmi a chiave pubblica, algoritmi di cifratura simmetrica, e funzioni hash. Esse possono essere usate insieme ai kit di sviluppo di SSL. Le principali implementazioni di SSL sono:

SSLRef: implementazione originale della Netscape.

SSLPlus: implementazione commerciale della Consensus con codice sorgente. È un ampliamento di SSLRef e richiede BSAFE3.0 della RSA.

SSLava: implementazione scritta in Java da parte della Phaos Technology.

SSLeay: implementazione non commerciale di SSL2.0 e SSL3.0. SSLeay implementa la crittografia a chiave pubblica con algoritmi usabili fuori dagli Stati Uniti, quindi non è soggetta alle restrizioni imposte al software di provenienza USA.

5 Creazione di una CA

La creazione di una CA deve tener conto di vari aspetti: supporto legale e amministrativo, progettazione e gestione. In questo articolo affronteremo soltanto le parti che riguardano la progettazione e la gestione; tralasceremo invece gli aspetti inerenti le problematiche legali e amministrative in quanto la legislazione italiana è ancora carente in materia.

5.1 Scelte progettuali

Il primo passo da affrontare dal punto di vista della progettazione è la scelta di strumenti adeguati sia alla gestione dei certificati, sia all'implementazione del protocollo SSL, sia alla creazione di un'interfaccia verso gli utenti.

Analizzando le soluzioni disponibili sul mercato, la nostra scelta è caduta sul pacchetto *SSLeay*, perché, oltre a soddisfare le nostre esigenze, è gratuito e non presenta i problemi legati all'esportazione di algoritmi crittografici fuori dagli USA.

Per l'interazione con gli utenti si è pensato di utilizzare l'interfaccia *Web* grazie all'utilizzo di un server Apache con il supporto di SSL: in particolare si sono scelte due valide implementazioni di tale server e cioè la versione di *Ben Laurie* chiamata *Apache-SSL* che è gratuita per scopi commerciali e non, e *Stronghold* della *C2NET Software Inc.* che è gratuito solo per scopi non commerciali e si basa sull'implementazione di *Ben Laurie*.

Il passo successivo è la scelta di una piattaforma hardware affidabile e di un sistema operativo efficiente. Valutando il parco macchine disponibile presso il Dipartimento di Informatica dell'Università di Salerno si è deciso di utilizzare un *Digital Alpha* con sistema operativo *Digital Unix*.

In un ambiente commerciale i vari compiti della CA devono essere separati e svolti da più macchine: in particolare, almeno una macchina *off-line* che gestisca i certificati, che sia schermata da intercettazioni elettromagnetiche, dotata di unità di backup e localizzata in locali ad accesso controllato; ed un'altra che si occupi dell'interfaccia verso gli utenti, che possa comunicare con la macchina *off-line* in modo sicuro, e che sia protetta con un sistema di sicurezza come un *firewall*.

5.2 Realizzazione di una CA usando SSLeay

D'ora in poi si suppone che il lettore abbia familiarità con sistemi operativi tipo UNIX perché per l'installazione di SSLeay ed il suo utilizzo si usano gli strumenti offerti dal sistema operativo UNIX (è possibile installare SSLeay anche su piattaforme Windows NT).

Il pacchetto SSLeay è disponibile su vari siti tra i quali il sito ufficiale `ftp://ftp.psy.uq.oz-.au/pub/Crypto/SSL/` ed il nostro mirror `ftp://cert.unisa.it/pub/Tools/Network/ssleay/`.

Le nostre considerazioni sono relative alla distribuzione `SSLeay-0.8.1.tar.gz`. Per l'installazione seguire le direttive contenute nel file INSTALL, attenendosi al PATH consigliato che è `/usr/local/ssl`.

Prima di utilizzare SSLeay bisogna creare un ambiente per la gestione dei certificati dando i seguenti comandi:

```
setenv SSLDIR /usr/local/ssl
mkdir ${SSLDIR}/certs
mkdir ${SSLDIR}/crl
mkdir ${SSLDIR}/newcerts
mkdir ${SSLDIR}/private
echo '01' > ${SSLDIR}/serial
touch ${SSLDIR}/index.txt
```

5.3 Il file di configurazione di SSLeay

Aggiornare il file di configurazione di SSLeay `/usr/local/ssl/lib/ssleay.cnf` in modo che soddisfi le proprie esigenze. Il file di configurazione contiene delle sezioni di configurazione specificate con delle etichette tra parentesi quadre. In questo modo, può essere utilizzato per vari scopi dai diversi comandi di SSLeay; tali sezioni devono essere modificate prima di qualunque altra operazione.

```
#
# SSLeay example configuration file.
# This is mostly being used for generation of certificate requests.
#

RANDFILE = $ENV::HOME/.rnd

####################################################################
[ ca ]
default_ca = CA_default # The default ca section

####################################################################
[ CA_default ]

dir = ./demoCA # Where everything is kept
certs = $dir/certs # Where the issued certs are kept
crl_dir = $dir/crl # Where the issued crl are kept
database = $dir/index.txt # database index file.
new_certs_dir = $dir/newcerts # default place for new certs.

certificate = $dir/cacert.pem  # The CA certificate
serial = $dir/serial  # The current serial number
crl = $dir/crl.pem  # The current CRL
private_key = $dir/private/cakey.pem# The private key
RANDFILE = $dir/private/.rand # private random number file

x509_extensions = x509v3_extensions # The extentions to add to the cert
default_days = 365 # how long to certify for
```

```
default_crl_days= 30 # how long before next CRL
default_md = md5 # which md to use.
preserve = no # keep passed DN ordering

# A few difference way of specifying how similar the request should look
# For type CA, the listed attributes must be the same, and the optional
# and supplied fields are just that :-)
policy = policy_match

####################################################################
# For the CA policy
[ policy_match ]
countryName = match
stateOrProvinceName = match
organizationName = match
organizationalUnitName = optional
commonName = supplied
emailAddress = optional

####################################################################
# For the 'anything' policy
# At this point in time, you must list all acceptable 'object'
# types.
[ policy_anything ]
countryName = optional
stateOrProvinceName = optional
localityName = optional
organizationName = optional
organizationalUnitName = optional
commonName = supplied
emailAddress = optional

####################################################################
[ req ]
default_bits = 1024
default_keyfile  = privkey.pem
distinguished_name = req_distinguished_name
attributes = req_attributes

####################################################################
[ req_distinguished_name ]
countryName = Country Name (2 letter code)
countryName_default = AU
countryName_min = 2
countryName_max = 2

stateOrProvinceName = State or Province Name (full name)
stateOrProvinceName_default = Some-State

localityName = Locality Name (eg, city)

0.organizationName = Organization Name (eg, company)
0.organizationName_default = Internet Widgits Pty Ltd

# we can do this but it is not needed normally :-)
#1.organizationName = Second Organization Name (eg, company)
```

```
#1.organizationName_default = CryptSoft Pty Ltd

organizationalUnitName = Organizational Unit Name (eg, section)
#organizationalUnitName_default =

commonName = Common Name (eg, YOUR name)
commonName_max = 64

emailAddress = Email Address
emailAddress_max = 40

####################################################################
[ req_attributes ]
challengePassword = A challenge password
challengePassword_min = 4
challengePassword_max = 20

unstructuredName = An optional company name

####################################################################
[ x509v3_extensions ]

nsCaRevocationUrl = http://www.cryptsoft.com/ca-crl.pem
nsComment = "This is a comment"

# under ASN.1, the 0 bit would be encoded as 80
nsCertType = 0x40

#nsBaseUrl
#nsRevocationUrl
#nsRenewalUrl
#nsCaPolicyUrl
#nsSslServerName
#nsCertSequence
#nsCertExt
#nsDataType
```

La prima sezione, etichettata con `[ ca ]`, contiene il nome della sezione contenente i parametri di default utilizzati dal comando `ca`. Ci possono essere anche più di una di queste sezioni e possono essere selezionate abilitando l'opzione `-name arg` del comando `ca`. Così, si ha la possibilità, di gestire più CA contemporaneamente.

La sezione etichettata con `[ req ]` indica le sezioni contenenti i parametri di default per il comando `req`, utilizzato per creare una richiesta di certificato. Se il pacchetto deve essere usato per gestire una CA allora queste sezioni devono contenere il valori di default per la creazione di certificati interni alla CA, come quelli di sottoCA dedicate a scopi particolari. Nel caso in cui il pacchetto viene usato insieme ad un server HTTP, allora i valori da impostare sono quelli che devono indicare i dati del server.

Vi sono inoltre delle sezioni di default per le politiche riguardo ai *Distinguished Name* dei certificati; esse specificano quali campi devono essere presenti (*supplied*) nella richiesta, quali campi devono essere presenti nella richiesta ed avere valori identici (*match*) a quelli del certificato della CA, e infine quali campi sono opzionali (*optional*). Anche per queste sezioni, se ne possono fare delle altre che possono poi essere utilizzate tramite l'opzione `-policy arg` del comando `ca`.

5.4 Creazione di un certificato self-signed

A questo punto bisogna creare un certificato autofirmato per la CA. Realizziamo questo passo con il comando `req` di SSLeay.

```
$SSLDIR/bin/ssleay req -new -x509 -keyout ${SSLDIR}/private/cakey.pem \
         -out ${SSLDIR}/private/cacert.pem \
 -config /usr/local/ssl/lib/ssleay.cnf
```

Questo comando produce un file contenente un certificato (*cacert.pem*) e un file contenente la chiave privata (*cakey.pem*). Il certificato della CA e i file contenenti le chiavi devono rimanere nella directory *$SSLDIR/private*, dove SSLeay li cercherà per default (come specificato dalla sezione *CA_default* di *ssleay.cnf*). Ora, quello che resta da fare è realizzare l'interfaccia con gli utenti. L'interfaccia si realizza tramite delle pagine HTML contenti le *form* per ricevere le richieste di certificato, per interrogare il database contente le informazioni sullo status di un certificato, per ricevere il certificato firmato e così via. Inoltre si devono realizzare anche gli script CGI che devono ricevere e fornire le informazioni. Tutta questa parte, ovviamente, è a discrezione di chi realizza l'interfaccia.

5.5 Firma di certificati

Una volta ricevuta una richiesta di firma di certificato in formato PEM (supponendo che si trovi nel file `request.pem`), la si può firmare utilizzando il comando `ca`:

```
$SSLDIR/bin/ssleay ca -policy policy_anything \
-out newcert.pem -config /usr/local/ssl/lib/ssleay.cnf -infiles request.pem
```

questo se si tratta di richiesta per un certificato server, utilizzabile con un server HTTP, un server FTP (*ftpd*), un server TELNET (*telnetd*). Il comando `ca` può specificare tramite l'opzione `-name arg` la sezione del file di configurazione da usare: si possono avere sezioni diverse per tipi di certificati diversi, per esempio una sezione *[CA_client]* per certificati di tipo client, una sezione *[CA_smime]* per certificati per S/MIME, una sezione *[CA_server]* per certificati di tipo server. Se invece si tratta di una richiesta di certificato client, si deve distinguere se il certificato deve essere utilizzato da un browser o da programmi di E-mail. Nel caso dei browser, vi sono distinzioni anche a seconda del browser. Infatti, per firmare un certificato per *Netscape Navigator* o *Netscape Communicator* bisogna utilizzare `ca` con le opzioni seguenti:

```
$SSLDIR/bin/ssleay ca -policy policy_anything \
-out newcert.der -config /usr/local/ssl/lib/ssleay.cnf -spkac request.pem
```

mentre per Internet Explorer la procedura è totalmente differente:

```
$SSLDIR/bin/ssleay ca -policy policy_anything -msie_hack\
-out temp.pem -config /usr/local/ssl/lib/ssleay.cnf -infiles request.pem

$SSLDIR/bin/ssleay crl2pkcs7 -certfile temp.pem -nocrl -out newcert.pem

rm temp.pem
```

5.6 Generazione di richieste di certificato

Per generare le richieste da inviare alla CA, i procedimenti sono diversi a seconda del tipo di certificato; se si vuole un certificato per un server e si vuole generare la richiesta con il pacchetto SSLeay, allora si deve dare il comando seguente:

```
$SSLDIR/bin/ssleay req -new -keyout serverkey.pem\
-out serverreq.pem -days 360 -config /usr/local/ssl/lib/ssleay.cnf
```

In questo modo viene creata una richiesta di certificato conforme allo standard *PKCS#10*. La chiave privata generata viene inserita nel file serverkey.pem e si deve conservare gelosamente, mentre la chiave pubblica insieme alla richiesta viene inserita nel file serverreq.pem. Il contenuto di quest'ultimo file deve essere inviato ad una CA per essere firmato. Il certificato restituito dalla CA, quindi, insieme alla chiave privata deve essere poi reso disponibile al server e questa operazione dipende dalla applicazione utilizzata. Ad esempio se si utilizza Apache+ssl il file di configurazione del Web server `httpd.conf` deve essere modificato in modo da indicare dove si trova il certificato del server e il file con le chiavi:

```
SSLCertificateFile /usr/local/ssl/certs/sitecert.pem
SSLCertificateKeyFile /usr/local/ssl/certs/sitekey.pem
```

dove sitekey.pem è una copia di serverkey.pem mentre sitecert.pem contiene il certificato fornito dalla CA. La creazione e l'installazione di un certificato client è più difficoltosa del creare un certificato server perché il client deve generare una coppia di chiavi, mantenere per se la chiave privata, mandare la chiave pubblica alla Certificate Authority che ingloberà il tutto in una richiesta di certificato (CSR). Quando la CA avrà firmato tale richiesta, il certificato client sarà disponibile e il client potrà installarlo nel browser in modo da poterlo presentare quando richiesto. Purtroppo non esiste un metodo unico per la creazione di certificati client: i prodotti Netscape utilizzano un meccanismo completamente diverso da quelli di Microsoft. La procedura per la creazione di un certificato client implica l'utilizzo di *form* HTML; queste form includono caratteristiche specifiche del client come speciali *tag* o programmi JavaScript, script CGI Perl che invocano il pacchetto *SSLeay*. Tali procedure non richiedono nessuna caratteristica da parte del server, tranne quella di eseguire script CGI. Il processo di creazione di un certificato client può essere completamente automatico, anche se in un ambiente produttivo c'è bisogno di validare la richiesta a seconda della politica di gestione dei certificati da parte della CA.

I passi principali per la creazione di un certificato client sono i seguenti:

- L'utente carica una pagina HTML nel quale inserisce i dati da inglobare nel certificato
- All'atto della sottomissione della form si verificano i seguenti eventi:
  - Il browser genera una coppia di chiavi (pubblica e privata)
  - La chiave privata viene memorizza nel browser
  - La chiave pubblica insieme alle informazioni di identità del richiedente, vengono inviate al server
  - Uno script CGI sul server crea il certificato e lo carica all'interno del client.

La form HTML include alcuni campi per le varie parti del *Distinguished Name* da inglobare nel certificato. Nel Netscape Navigator (Communicator) la form contiene un tag aggiuntivo `<KEYGEN>`. Questo tag crea la coppia di chiavi, e fa si che la chiave pubblica venga restituita come valore di un campo della form che viene inviata. Il tag `<KEYGEN>` visualizza la scelta (*select*) tra vari gradi di sicurezza per la grandezza delle chiavi: nella versioni adatte all'esportazione viene visualizzata solo l'unica scelta di 512 bit, mentre le altre versioni presentano la scelta tra 512 , 768 e 1024 bit. Tutte queste informazioni vengono utilizzate dallo script CGI per formare una richiesta di certificato, e questa richiesta viene usata per creare il certificato client.

La form HTML per i prodotti Microsoft è più complicata perché richiede l'uso di uno script JavaScript nella pagina per utilizzare una DLL *ActiveX* per la generazione della coppia di chiavi e la richiesta di certificato. Lo script JavaScript viene scaricato insieme alla pagina HTML e viene chiamato all'atto della *Submit* della form. Il programma chiama il metodo *GenReqForm* del controllo ActiveX di nome *certenr3.dll*, passandogli i valori del Distinguished Name della form. La richiesta di certificato così creata viene caricata in un campo hidden della form e passata ad uno script CGI del server.

Sia Netscape Navigator che Microsoft Internet Explore presenta una sequenza di dialogo come parte della creazione della coppia di chiavi. Se tale sequenza si completa con successo, il server invocherà lo script corrispondente con i valori passati dalla form. Lo script CGI per Netscape Navigator crea un file contenente i valori del Distinguished Name ritornati dalla form, e un campo speciale di nome *SPKAC* (Signed Public Key And Challenge). Questo file sarà in seguito firmato dalla CA che segnalerà all'utente un URL per prelevare il certificato. L'utente si collega alla pagina segnalata ed esegue il download, il server chiama uno script CGI che invia il certificato firmato con un Content-Type *application/x-509-user-cert*. Il Navigator riconosce questo particolare tipo e richiede all'utente di installare il certificato client nel browser.

Lo script CGI per processare le richieste fatte da Internet Explorer è molto più complicato. Prende la richiesta di certificato ritornata dalla form e la riformatta su linee di 72 caratteri; creando una richiesta che in seguito sarà in seguito firmato dalla CA che segnalerà all'utente un URL per prelevare il certificato. L'utente si collega alla pagina segnalata ed esegue il dowbload, il server chiama uno script CGI che crea dinamicamente una pagina HTML che contiene un JavaScripti per chiamare il metodo *AcceptCredentials* del controllo ActiveX *certenr3.dll*, che a sua volta installa il certificato nel formato *PKCS#7*.

```
 1 % s_client -verify 3 -cert cert_client.pem -key cert_client_key.pem -CAf
ile root_ca_unisa_cert.pem -state -ssl3
 2 verify depth is 3
 3 CONNECTED(00000003)
 4 SSL_connect:before SSL initalisation
 5 SSL_connect:SSLv3 write client hello A
 6 SSL_connect:SSLv3 read server hello A
 7 depth=1 /C=IT/ST=Salerno/L=Baronissi/O=Security Service Group/O=Univ. di
 Salerno - Dip. di Informatica ed Applicazioni/OU=SSG Certificate Authority
/CN=SSG Root CA Certificate/Email=info@cert.unisa.it
 8 verify return:1
 9 depth=0 /C=IT/ST=Salerno/L=Baronissi/O=SSG Secure Server/OU=Security Ser
vice Group CA/CN=cert.unisa.it
10 verify return:1
11 SSL_connect:SSLv3 read server certificate A
12 SSL_connect:SSLv3 read server certificate request A
13 SSL_connect:SSLv3 read server done A
14 SSL_connect:SSLv3 write client certificate A
15 SSL_connect:SSLv3 write client key exchange A
16 SSL_connect:SSLv3 write certificate verify A
17 SSL_connect:SSLv3 write change cipher spec A
18 SSL_connect:SSLv3 write finished A
19 SSL_connect:SSLv3 flush data
20 SSL_connect:SSLv3 read finished A
   ---
   Certficate chain
    0 s:/C=IT/ST=Salerno/L=Baronissi/O=SSG Secure Server/OU=Security Servic
e Group CA/CN=cert.unisa.it
      i:/C=IT/ST=Salerno/L=Baronissi/O=Security Service Group/O=Univ. di Sa
lerno - Dip. di Informatica ed Applicazioni/OU=SSG Certificate Authority/CN
=SSG Root CA Certificate/Email=info@cert.unisa.it
    1 s:/C=IT/ST=Salerno/L=Baronissi/O=Security Service Group/O=Univ. di Sa
lerno - Dip. di Informatica ed Applicazioni/OU=SSG Certificate Authority/CN
=SSG Root CA Certificate/Email=info@cert.unisa.it
      i:/C=IT/ST=Salerno/L=Baronissi/O=Security Service Group/O=Univ. di Sa
lerno - Dip. di Informatica ed Applicazioni/OU=SSG Certificate Authority/CN
=SSG Root CA Certificate/Email=info@cert.unisa.it
   ---
   Acceptable client certificate CA names
   /C=IT/ST=Salerno/L=Baronissi/O=SSG Secure Server/OU=Security Service Gro
up CA/CN=cert.unisa.it
   ---
   SSL handshake has read 2325 bytes and written 2451 bytes
   ---
   New, SSLv3, Cipher is RC4-SHA
   SSL-Session:
       Cipher    : RC4-SHA
       Session-ID: E4FACE57AD722196593A0BFE7D1BE63E10234E9C4CF6F1A239373BFE
EB77377B
       Master-Key: CEF5F561894A1A363F32B051B548717A12419D39FB230D2F4608FDA2
B0FB0946ADD192F6BD110C29878F43DC1E39AB70
       Key-Arg   : None
       Start Time: 888229584
       Timeout   : 7200 (sec)
   ---
   GET / HTTP/1.0
   HTTP/1.0 200 ok
   Content-type: text/html

   <HTML>
   <HEAD>
   <TITLE>Test Page</TITLE>
   </HEAD>
   <BODY>
   <CENTER><H1>Test Page</H1></CENTER>
   </BODY>
   </HTML>
   SSL3 alert read:warning:close notify
   closed
   SSL3 alert write:warning:close notify
```

Figura 6: Tracciato client *s_client*