7.2 Secure socket layer (SSL) e transport layer security (TLS)

SSL è stato creato da Netscape. La versione 3 del protocollo è stata progettata con una revisione pubblica e con fonti di provenienza aziendale ed è stata pubblicata come documento Internet draft. Successivamente, una volta raggiunto il consenso a sottomettere il protocollo al processo di standardizzazione Internet, all'interno di IETF si costituì il gruppo di lavoro TLS per lo sviluppo di uno standard comune. Le attività in corso relativamente a TLS sono rivolte alla creazione di una versione iniziale di un documento Internet standard. Si può considerare questa prima versione di TLS essenzialmente come un SSLv3.1, molto vicino e compatibile con SSLv3.

Il fulcro di questo paragrafo concerne la presentazione di SSLv3 e si conclude con la descrizione delle differenze fondamentali fra SSLv3 e TLS.

Architettura SSL

SSL è progettato per far uso del protocollo TCP al fine di fornire un servizio di sicurezza end-to-end affidabile. Come mostra la Figura 7.2, SSL non è un unico protocollo, ma è piuttosto costituito da due livelli di protocolli.

Il protocollo SSL Record fornisce servizi di sicurezza di base a un certo numero di protocolli di più alto livello. In particolare, il protocollo HTTP (*hypertext transfer protocol*), definito nel documento RFC 2068, che fornisce il servizio di trasferimento per le interazioni client/server web, può operare sopra SSL. SSL contiene tre protocolli di più alto livello: il protocollo Handshake, il protocollo Change cipher spec e il protocollo Alert. Questi protocolli specifici di SSL vengono utilizzati nella gestione degli scambi SSL e sono presi in esame alla fine di questo paragrafo.

Due concetti importanti di SSL sono quello di connessione SSL e di sessione.

- **Connessione.** Una connessione è un trasporto (nella definizione del modello a strati OSI) che fornisce una tipologia opportuna di servizio. Per SSL, queste connessioni sono relazioni peer-to-peer. Le connessioni sono transitorie. Ogni connessione è associata a una sessione.

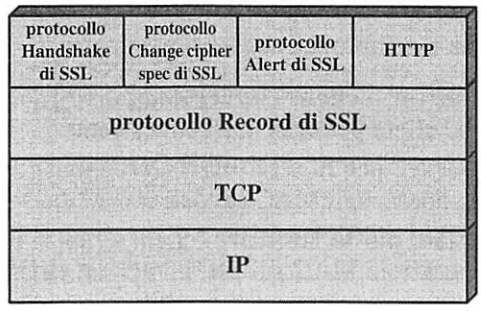

Figura 7.2 *Pila del protocollo SSL.*

- **Sessione.** Una sessione SSL è un'associazione fra un client e un server. Le sessioni sono create dal protocollo Handshake e definiscono un insieme di parametri crittografici di sicurezza che possono essere condivisi fra molteplici connessioni. Le sessioni si utilizzano per evitare una costosa negoziazione di nuovi parametri di sicurezza per ciascuna connessione.

Ci possono essere molteplici connessioni sicure fra una qualunque coppia di parti (applicazioni come HTTP su client e server). In teoria ci potrebbero essere anche molteplici sessioni simultanee fra due parti, ma questa funzionalità non è utilizzata nella pratica.

A ciascuna sessione è associato un certo numero di stati. Una volta instaurata una sessione, c'è uno stato operativo corrente sia per la lettura che per la scrittura (per esempio, ricezione e trasmissione). Inoltre, durante il protocollo Handshake vengono creati gli stati di attesa (*pending*) per lettura e scrittura. Alla positiva conclusione del protocollo Handshake, gli stati di attesa diventano stati correnti.

Lo stato di una sessione è definito dai seguenti parametri (le cui definizioni sono tratte dalla specifica SSL).

- **Identificatore di sessione** (*session identifier*): sequenza arbitraria di byte scelta dal server per identificare uno stato di sessione attivo o riattivabile.
- **Certificato del peer** (*peer certificate*): certificato X509.v3 del peer. Questo elemento dello stato può essere nullo.
- **Metodo di compressione** (*compression method*): algoritmo usato per la compressione dei dati prima della cifratura.
- **Specifica del cifrario** (*cipher spec*): specifica l'algoritmo di cifratura dei dati che viene utilizzato – come, ad esempio, nessun algoritmo (null), DES, etc. – e l'algoritmo hash (ad esempio, MD5 o SHA-1) utilizzato per il calcolo del MAC. Definisce anche gli attributi crittografici come, ad esempio, la dimensione del codice hash (hash_size).
- **Segreto principale** (*master secret*): valore segreto a 48 byte condiviso fra il client e il server.
- **È riattivabile** (*is resumable*): flag che indica se la sessione può essere usata per instaurare nuove connessioni.

Uno stato di connessione è definito dai seguenti parametri.

- **Valore casuale di server e client** (*server and client random*): sequenze di byte scelte dal server e dal client per ciascuna connessione.
- **MAC segreto del server per la scrittura** (*server write MAC secret*): chiave segreta utilizzata nelle operazioni MAC sui dati inviati dal server.
- **MAC segreto del client per la scrittura** (*client write MAC secret*): chiave segreta utilizzata nelle operazioni MAC sui dati inviati dal client.
- **Chiave di scrittura del server** (*server write key*): chiave di cifratura convenzionale per i dati cifrati dal server e decifrati dal client.

- **Chiave di scrittura del client** (*client write key*): chiave di cifratura convenzionale per i dati cifrati dal client e decifrati dal server.
- **Vettori di inizializzazione** (*initialization vectors*): quando si utilizza un cifrario a blocchi in modalità CBC, viene mantenuto un vettore di inizializzazione (IV) per ciascuna chiave. Tale campo è dapprima inizializzato dal protocollo Handshake di SSL. Successivamente, l'ultimo blocco di testo cifrato di ciascun record viene mantenuto per essere utilizzato come IV per il record successivo.
- **Numeri di sequenza** (*sequence numbers*): ciascuna parte mantiene, per ciascuna connessione, numeri di sequenza separati per i messaggi trasmessi e ricevuti. Quando una parte invia o riceve un messaggio change cipher spec per la modifica della specifica del cifrario, si pone a zero il numero di sequenza appropriato. I numeri di sequenza non possono superare il valore $2^{64} - 1$.

Protocollo Record di SSL

Il protocollo Record di SSL fornisce i servizi di riservatezza e integrità dei messaggi per le connessioni SSL.

La Figura 7.3 riporta il funzionamento complessivo del protocollo Record di SSL. Il protocollo Record prende un messaggio applicativo da trasmettere, effettua la frammentazione dei dati in blocchi più piccoli, eventualmente esegue la compressione dei dati, applica il MAC, esegue la cifratura, aggiunge un'intestazione e trasmette il messaggio risultante in un segmento TCP. La chiave segreta condivisa utilizzata per la cifratura convenzionale dei payload di SSL e quella utilizzata per creare il codice di autenticazione del messaggio (MAC) sono definite tramite il protocollo Handshake. I dati

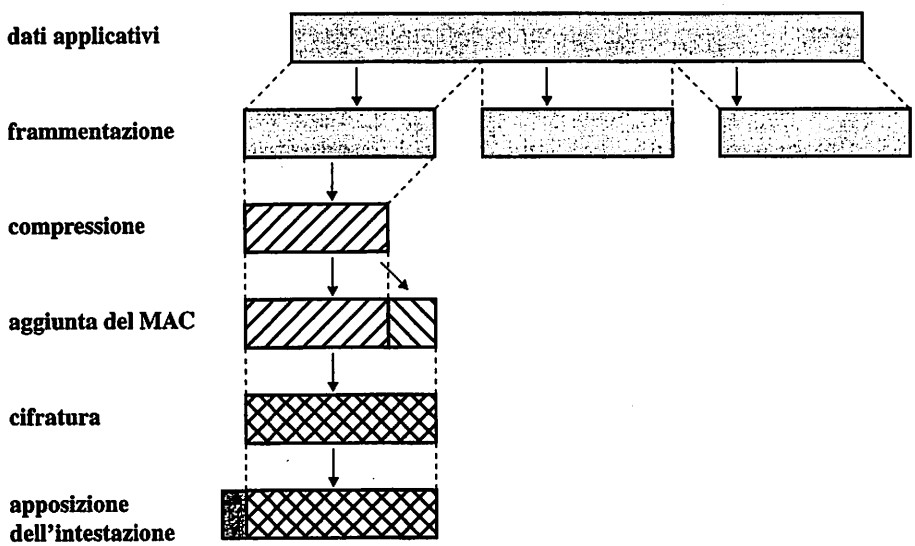

Figura 7.3 *Funzionamento del protocollo Record di SSL.*

ricevuti sono decifrati, verificati, decompressi, riassemblati e quindi consegnati agli utenti a livello più alto.

Il primo passo è la **frammentazione**. Ogni messaggio dei livelli sovrastanti viene frammentato in blocchi di dimensione 2^{14} byte (16384 byte) o di dimensione inferiore. Successivamente e facoltativamente si applica la **compressione**. La compressione deve essere senza perdita e non dovrebbe aumentare la lunghezza del contenuto di più di 1024 byte.[2] In SSLv3 (come anche nella versione corrente di TLS), non è specificato alcun algoritmo di compressione, quindi il valore di default per l'algoritmo di compressione è null.

Il passo di elaborazione successivo consiste nel calcolare un MAC sui dati compressi. A tale scopo viene utilizzata una chiave segreta condivisa. Il calcolo del MAC è definito come segue:

$$\text{hash}(\text{MAC\_write\_secret} \| \text{pad\_2} \| \text{hash}(\text{MAC\_write\_secret} \| \text{pad\_1} \| \text{seq\_num} \| \text{SSLCompressed.type} \| \text{SSLCompressed.length} \| \text{SSLCompressed.fragment}))$$

dove

$\|$	= operatore di concatenazione
MAC_write_secret	= chiave segreta condivisa
hash	= algoritmo hash crittografico (MD5 o SHA-1)
pad_1	= il byte 0 × 36 (0011 0110) ripetuto 48 volte (384 bit) per MD5 e 40 volte (320 bit) per SHA-1
pad_2	= il byte 0 × 5C (0101 1100) ripetuto 48 volte (384 bit) per MD5 e 40 volte (320 bit) per SHA-1
seq_num	= numero di sequenza per il messaggio considerato
SSLCompressed.type	= protocollo di livello più alto utilizzato per elaborare il frammento considerato
SSLCompressed.length	= lunghezza del frammento compresso
SSLCompressed.fragment	= frammento compresso (se non si utilizza la compressione, il frammento è in chiaro)

Come si può notare, l'algoritmo appena descritto è molto simile all'algoritmo HMAC definito nel Capitolo 3. La differenza è che in SSLv3 i due pad sono concatenati mentre in HMAC sono combinati mediante OR esclusivo. L'algoritmo MAC di SSLv3 è basato sull'Internet draft originale di HMAC, in cui si usava la concatenazione. La versione finale di HMAC, definita nel documento RFC 2104, usa l'OR esclusivo.

Il messaggio compresso e il MAC sono quindi **cifrati** attraverso cifratura simmetrica. La cifratura non dovrebbe aumentare la lunghezza del contenuto di più di 1024 byte,

[2] Ovviamente, si spera che la compressione riduca e non espanda i dati. Tuttavia, per blocchi molto piccoli, può verificarsi quest'ultima eventualità a causa delle convenzioni di formattazione.

in modo tale che la lunghezza totale non dovrebbe essere superiore a 2^{14} + 2048. Gli algoritmi di cifratura consentiti sono i seguenti:

Cifratura a blocchi		Cifratura a flusso	
Algoritmo	**Dimensioni della chiave**	**Algoritmo**	**Dimensioni della chiave**
IDEA	128	RC4-40	40
RC2-40	40	RC4-128	128
DES-40	40		
DES	56		
3DES	168		
Fortezza	80		

Per la cifratura a flusso, vengono cifrati il messaggio compresso e il MAC associato. Si noti che il MAC è calcolato prima di eseguire la cifratura e quindi viene cifrato insieme al testo in chiaro o con il testo in chiaro compresso.

Per la cifratura a blocchi, il completamento può essere aggiunto dopo il calcolo del MAC e prima della cifratura. Il completamento consiste in un certo numero di byte aggiuntivi seguiti da un byte che specifica la lunghezza del completamento stesso. La quantità totale del completamento corrisponde alla più piccola quantità tale che la dimensione totale dei dati da cifrare (testo in chiaro più MAC più completamento) sia un multiplo della lunghezza del blocco del cifrario. Un esempio potrebbe essere un testo in chiaro (o compresso, se viene utilizzata la compressione) di 58 byte con un MAC di 20 byte (impiegando SHA-1) cifrati utilizzando una lunghezza di blocco di 8 byte (per esempio DES). Contando anche il byte padding.length, si arriva a un totale di 79 byte. Per rendere il valore totale un intero multiplo di 8, si aggiunge un byte di completamento.

L'ultimo passo dell'elaborazione del protocollo Record di SSL consiste nell'apporre un'intestazione composta dai seguenti campi.

- **Tipo di contenuto, 8 bit** (*content type*): il protocollo di livello più alto utilizzato per elaborare il frammento accluso.
- **Versione principale, 8 bit** (*major version*): indica la versione principale di SSL in uso. Nel caso di SSLv3, il valore è 3.
- **Versione minore, 8 bit** (*minor version*): indica la versione minore di SSL in uso. Nel caso di SSLv3, il valore è 0.
- **Lunghezza compressa, 16 bit** (*compressed length*): lunghezza in byte del frammento in chiaro (o del frammento compresso se si usa la compressione). Il valore massimo è 2^{14} + 2048.

I tipi di contenuto definiti sono change_cipher_spec, alert, handshake, e application_data. I primi tre sono i protocolli specifici di SSL, descritti in seguito. Si noti che non vi è distinzione fra le diverse applicazioni che possono utilizzare SSL (per esempio HTTP); il contenuto dei dati generati da queste applicazioni non è noto a SSL.

La Figura 7.4 mostra il formato generato dal protocollo Record di SSL.

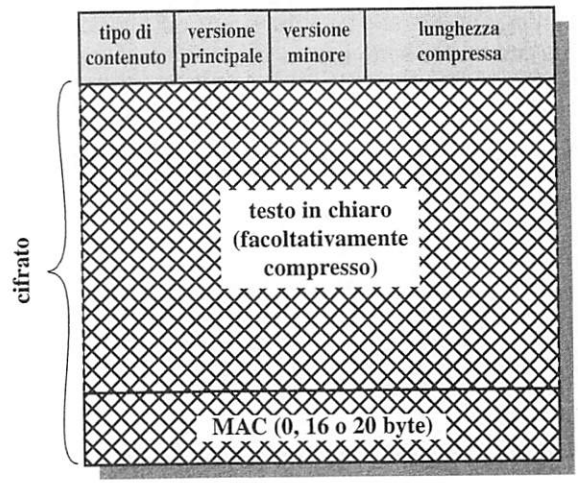

Figura 7.4 *Formato record di SSL.*

Protocollo Change cipher spec

Il protocollo Change cipher spec è il più semplice fra i tre specifici protocolli di SSL che utilizzano il protocollo Record di SSL. Consiste in un singolo messaggio (Figura 7.5a), che a sua volta consiste in un solo byte con valore 1. L'unico scopo di questo messaggio è quello di forzare la copiatura dello stato di attesa nello stato corrente, aggiornando la suite di cifratura da usare nella connessione corrente.

Protocollo Alert

Il protocollo Alert è utilizzato per comunicare al peer messaggi di allarme relativi a SSL. Come per le altre applicazioni che utilizzano SSL, i messaggi di allarme sono compressi e cifrati, come specificato nello stato corrente.

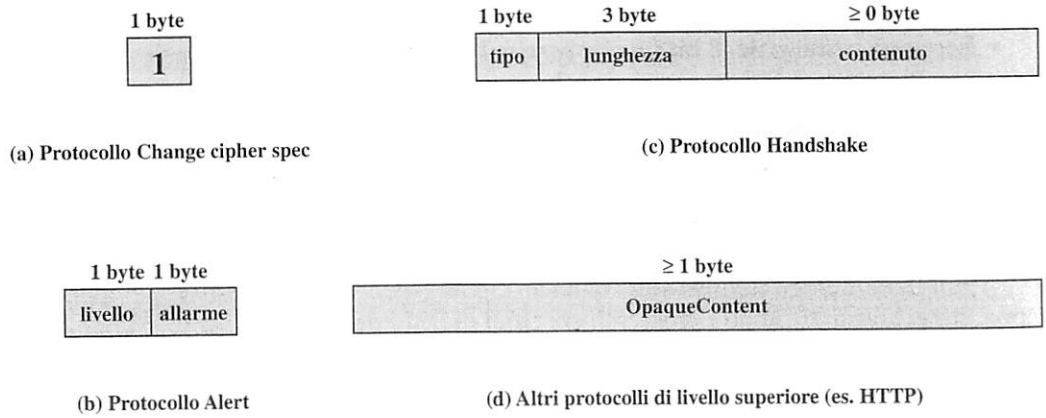

Figura 7.5 *Payload del protocollo Record di SSL.*

Ogni messaggio in questo protocollo è composto da due byte (Figura 7.5b). Il primo byte assume il valore warning(1) o fatal(2) per trasmettere la gravità del messaggio. Se il livello è fatal, SSL termina immediatamente la connessione. Le altre connessioni nella stessa sessione possono continuare, ma non si possono stabilire nuove connessioni nella sessione in questione. Il secondo byte contiene un codice che denota lo specifico allarme. Prima di tutto, vengono elencati gli allarmi che sono sempre fatali (le definizioni sono tratte dalla specifica SSL).

- **unexpected_message:** è stato ricevuto un messaggio non riconosciuto.
- **bad_record_mac:** è stato ricevuto un MAC errato.
- **decompression_failure:** la funzione di decompressione ha ricevuto un input improprio (per esempio, risulta incapace di effettuare la decompressione oppure effettua la decompressione a una lunghezza superiore a quella massima consentita).
- **handshake_failure:** il mittente non è stato capace di negoziare un insieme accettabile di parametri di sicurezza con le opzioni disponibili.
- **illegal_parameter:** un campo di un messaggio handshake conteneva un valore al di fuori dell'insieme dei valori ammissibili o era inconsistente con gli altri campi.

Gli allarmi rimanenti sono:

- **close_notify:** notifica al ricevente che il mittente non invierà più messaggi nella connessione corrente. Si richiede a ciascuna parte di inviare un allarme close_notify prima di chiudere la parte di scrittura di una connessione.
- **no_certificate:** può essere inviato in risposta a una richiesta di certificato quando non è disponibile alcun certificato adatto.
- **bad_certificate:** il certificato ricevuto era corrotto (per esempio, conteneva una firma non verificabile).
- **unsupported_certificate:** il certificato ricevuto ha un tipo non supportato.
- **certificate_revoked:** il certificato è stato revocato dal suo firmatario.
- **certificate_expired:** il certificato è scaduto.
- **certificate_unknown:** è intervenuto qualche altro problema, non noto durante l'elaborazione del certificato, che lo ha reso inaccettabile.

Protocollo Handshake

Il protocollo Handshake è la parte più complessa di SSL. Questo protocollo consente al client e al server di autenticarsi a vicenda e di negoziare un algoritmo di cifratura e di MAC e le chiavi di cifratura da utilizzare per la protezione dei dati inviati in un record SSL (cfr Figura 7.4).

Il protocollo Handshake è utilizzato prima di spedire qualunque dato applicativo. Consiste in una serie di messaggi scambiati fra client e server. Tutti questi messaggi hanno il formato mostrato nella Figura 7.5c. Ciascun messaggio ha tre campi.

- **Tipo, 1 byte** (*type*): indica un tipo fra 10 possibili tipi di messaggio, riportati nella Tabella 7.2.

Tabella 7.2 *Tipi di messaggio del protocollo Handshake di SSL*

Tipo di messaggio	Parametri
hello_request	Nessun parametro
client_hello	Versione, random, ID di sessione, suite di cifratura, metodo di compressione
server_hello	Versione, random, ID di sessione, suite di cifratura, metodo di compressione
certificate	Catena di certificati X.509v3
server_key_exchange	Parametri, firma
certificate_request	Tipo, autorità
server_done	Nessun parametro
certificate_verify	Firma
client_key_exchange	Parametri, firma
finished	Valore hash

- **Lunghezza, 3 byte** (*length*): lunghezza del messaggio in byte.
- **Contenuto, ≥1 byte** (*content*) parametri associati con il messaggio in questione, elencati nella Tabella 7.2.

La Figura 7.6 mostra lo scambio iniziale necessario a instaurare una connessione logica fra client e server. Lo scambio può essere suddiviso in quattro fasi.

Fase 1. Stabilire le capacità di sicurezza

Questa fase è usata per instaurare una connessione logica e per stabilire le capacità di sicurezza che saranno ad essa associate. Il client inizia lo scambio inviando un **messaggio client_hello** con i seguenti parametri.

- **Versione** (*version*): la versione più elevata di SSL supportata dal client.
- **Random:** struttura casuale generata dal client, che consiste in un timestamp a 32 bit e in 28 byte prodotti da un generatore di numeri casuali sicuro. Tali valori servono come nonce e sono utilizzati durante lo scambio delle chiavi per la prevenzione di attacchi di replay.
- **ID di sessione** (*session ID*): identificatore di sessione a lunghezza variabile. Un valore diverso da zero denota che il client vuole aggiornare i parametri di una connessione esistente o creare una nuova connessione nella sessione attuale. Un valore zero indica che il client vuole instaurare una nuova connessione in una nuova sessione.
- **Suite di cifratura** (*CipherSuite*): lista contenente le combinazioni di algoritmi crittografici supportati dal client, in ordine decrescente di preferenza. Ciascun elemento della lista (ovvero ciascuna suite di cifratura) definisce sia l'algoritmo per lo scambio delle chiavi sia una CipherSpec (entrambi descritti in seguito).
- **Metodo di compressione** (*compression method*): lista dei metodi di compressione supportati dal client.

Dopo aver inviato il messaggio client_hello, il client attende il **messaggio server_hello**, contenente gli stessi parametri del messaggio client_hello. Al messaggio

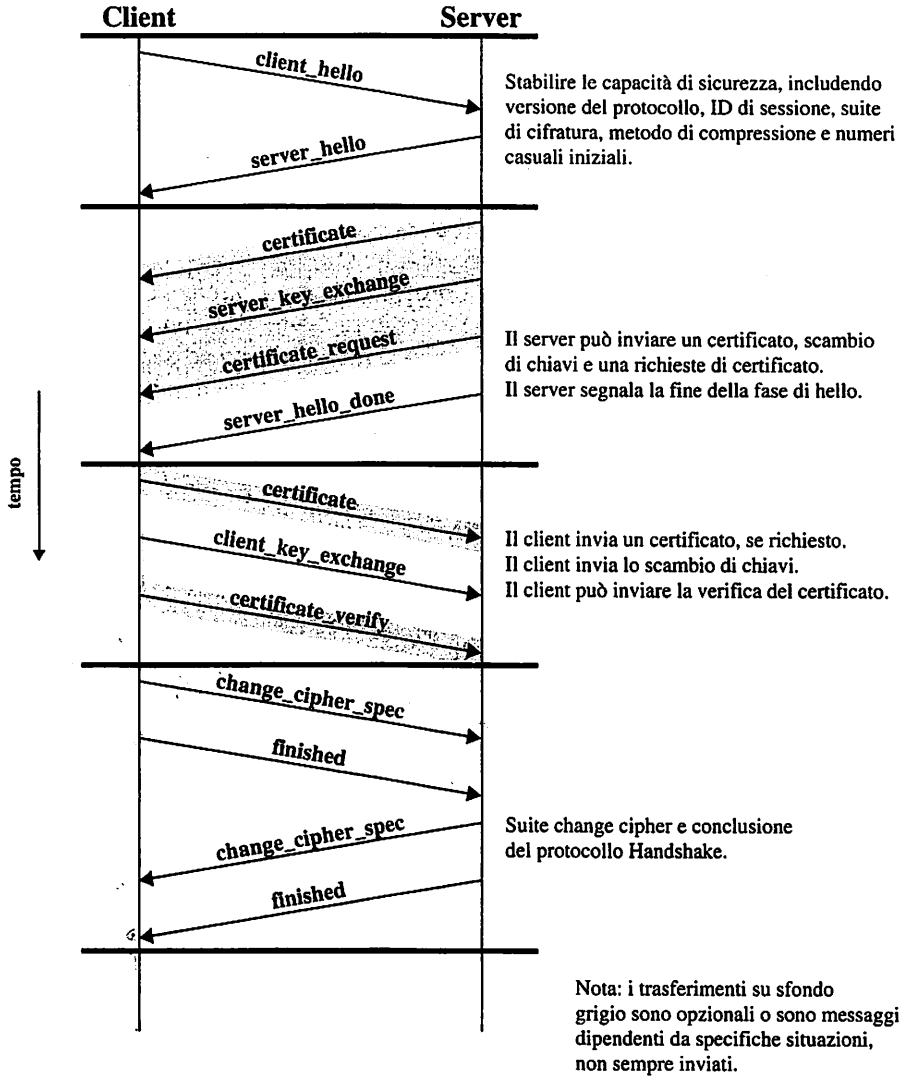

Figura 7.6 *Funzionamento del protocollo Handshake.*

server_hello si applicano le seguenti convenzioni. Il campo Versione contiene il valore di versione più basso suggerito dal client e il più elevato supportato dal server. Il campo Random viene generato dal server ed è indipendente dal campo Random del client. Se il campo ID di sessione del client era diverso da zero, lo stesso valore è usato dal server; altrimenti il campo ID di sessione del server contiene il valore di una nuova sessione. Il campo Suite di cifratura contiene una specifica suite di cifratura scelta dal

server fra quelle proposte dal client. Il campo Compressione contiene il metodo di compressione scelto dal server fra quelli proposti dal client. Il primo elemento del parametro Suite di cifratura è il metodo di scambio di chiavi (per esempio, lo strumento attraverso cui vengono scambiate le chiavi crittografiche per la cifratura convenzionale e il MAC). I metodi di scambio delle chiavi supportati sono i seguenti.

- **RSA.** La chiave segreta è cifrata con la chiave pubblica RSA del ricevente. Deve essere reso disponibile un certificato a chiave pubblica per la chiave del ricevente.
- **Fixed Diffie-Hellman.** Algoritmo di scambio di chiavi Diffie-Hellman in cui il certificato del server contiene i parametri pubblici Diffie-Hellman firmati dall'autorità certificativa (CA). Il client fornisce i propri parametri di chiave pubblica Diffie-Hellman all'interno di un certificato se è richiesta l'autenticazione del client, oppure in un messaggio di scambio di chiavi. Questo metodo porta ad avere una chiave segreta stabilita fra due parti, basata sul calcolo Diffie-Hellman utilizzando le chiavi pubbliche fissate.
- **Ephemeral Diffie-Hellman.** Tecnica utilizzata per generare chiavi segrete temporanee (one-time). In questo caso, vengono scambiate le chiavi pubbliche Diffie-Hellman firmate con la chiave privata RSA o DSS del mittente. Il ricevente può utilizzare la chiave pubblica corrispondente per verificare la firma. I certificati sono utilizzati per autenticare le chiavi pubbliche. Questa sembrerebbe la più sicura delle tre opzioni Diffie-Hellman in quanto genera una chiave temporanea e autenticata.
- **Anonymous Diffie-Hellman.** Utilizza l'algoritmo Diffie-Hellman di base, senza autenticazione. Ovvero, ciascuna parte invia i propri parametri pubblici all'altra parte, senza autenticazione. Questo approccio è vulnerabile rispetto ad attacchi di tipo man-in-the-middle, in cui l'avversario esegue l'algoritmo anonymous Diffie-Hellman con entrambe le parti.
- **Fortezza.** Tecnica definita per lo schema Fortezza.

La definizione del metodo per lo scambio delle chiavi è la CipherSpec che comprende i seguenti campi.

- **CipherAlgorithm** (algoritmo di cifratura): qualunque algoritmo precedentemente menzionato (RC4, RC2, DES, 3DES, DES40, IDEA, Fortezza).
- **MACAlgorithm** (algoritmo MAC): MD5 o SHA-1.
- **CipherType** (tipo di cifratura): a flusso o a blocchi.
- **IsExportable** (è esportabile): vero o falso.
- **HashSize** (dimensione hash): 0, 16 (per MD5) o 20 (per SHA-1) byte.
- **Key Material** (dati per le chiavi): sequenza di byte contenente i dati usati nella generazione delle chiavi di scrittura.
- **IV Size** (dimensione di IV): dimensione del vettore di inizializzazione per la cifratura in modalità CBC.

Fase 2. Autenticazione del server e scambio delle chiavi

Il server inizia questa fase inviando il proprio certificato, nel caso debba essere autenticato; il messaggio contiene un certificato X.509 o una catena di certificati. Il **mes-**

saggio certificate è richiesto per qualunque metodo di scambio delle chiavi concordato, ad eccezione del metodo anonymous Diffie-Hellman. Si noti che se si usa fixed Diffie-Hellman, il messaggio certificate ha il ruolo di messaggio server_key exchange in quanto comprende i parametri pubblici Diffie-Hellman del server.

Successivamente, si può inviare, se richiesto, un **messaggio server_key_exchange**. Non è richiesto in due casi: quando il server ha inviato un certificato con i parametri fixed Diffie-Hellman oppure quando si utilizza lo scambio di chiavi RSA. Il messaggio server_key_exchange è necessario nei seguenti casi.

- **Anonymous Diffie-Hellman.** Il contenuto del messaggio è composto dai due valori globali Diffie-Hellman (un numero primo e una radice primitiva del numero) e dalla chiave pubblica Diffie-Hellman del server (si veda la Figura 3.10).
- **Ephemeral Diffie-Hellman.** Il contenuto del messaggio comprende i tre parametri Diffie-Hellman forniti per anonymous Diffie-Hellman e la firma di questi parametri.
- **Scambio di chiavi RSA, in cui il server usa RSA ma ha una chiave RSA di sola firma.** Conformemente, il client non può semplicemente inviare una chiave segreta cifrata con la chiave pubblica del server. Al contrario, il server deve creare una coppia temporanea chiave pubblica/chiave privata RSA e utilizzare il messaggio server_key_exchange per inviare la chiave pubblica. Il messaggio comprende i due parametri della chiave pubblica RSA temporanea (esponente e modulo, cfr Figura 3.8) e una firma di questi parametri.
- **Fortezza.**

Come di consueto, una firma è creata considerando il codice hash di un messaggio e cifrandolo con la chiave privata del mittente. In questo caso, il codice hash è definito come:

```
hash (ClientHello.random || ServerHello.random || ServerParams)
```

Quindi il codice hash copre non solo i parametri Diffie-Hellman o RSA, ma anche i due nonce dei due messaggi hello iniziali, fornendo quindi garanzie contro attacchi di replay. In caso di una firma DSS, il codice hash è calcolato mediante l'algoritmo SHA-1. Nel caso di una firma RSA, vengono calcolati i valori hash sia MD5 che SHA-1, e la loro concatenazione (36 byte) viene cifrata con la chiave privata del server.

Successivamente, un server che non usa anonymous Diffie-Hellman può richiedere un certificato al client. Il **messaggio certificate_request** comprende due parametri: certificate_type (tipo di certificato) e certificate_authorities (autorità del certificato). Il tipo di certificato indica l'algoritmo a chiave pubblica e il suo impiego:

- RSA, solo firma
- DSS, solo firma
- RSA per fixed Diffie-Hellman (in questo caso la firma è usata solo per scopi di autenticazione, inviando un certificato firmato con RSA)
- DSS per fixed Diffie-Hellman (ugualmente usato solo per scopi di autenticazione)
- RSA per ephemeral Diffie-Hellman

- DSS per ephemeral Diffie-Hellman
- Fortezza

Il secondo parametro nel messaggio certificate_request è una lista di nomi di autorità certificative accettabili.

L'ultimo messaggio della Fase 2, sempre richiesto, è il **messaggio server_done**, inviato dal server per segnalare il completamento del server hello e dei messaggi associati. Dopo aver inviato questo messaggio, il server attende una risposta dal client. Questo messaggio non ha parametri.

Fase 3. Autenticazione del client e scambio delle chiavi

Alla ricezione del messaggio server_done, il client dovrebbe verificare che il server abbia fornito un certificato valido, se richiesto, e controllare che i parametri del messaggio server_hello siano accettabili. Se tutte queste condizioni sono soddisfatte, il client invia uno o più messaggi al server.

Se il server ha richiesto un certificato, il client inizia questa fase inviando un **messaggio certificate**. Il client invia invece un messaggio di allarme no_certificate qualora non fosse disponibile alcun certificato adatto.

Successivamente, deve essere inviato il messaggio **client_key_exchange**. Il contenuto del messaggio dipende dalla tipologia di scambio di chiavi, come segue.

- **RSA.** Il client genera un **valore segreto pre-master** di 48 byte e lo cifra con la chiave pubblica contenuta nel certificato del server o con la chiave RSA temporanea contenuta nel messaggio server_key_exchange. Il suo utilizzo nel calcolo del **valore master segreto** verrà spiegato in seguito.
- **Ephemeral o anonymous Diffie-Hellman.** Vengono inviati i parametri pubblici Diffie-Hellman del client.
- **Fixed Diffie-Hellman.** I parametri pubblici Diffie-Hellman del client sono stati inviati in un messaggio certificate, quindi il contenuto di questo messaggio è nullo.
- **Fortezza.** Sono inviati i parametri Fortezza del client.

Infine, in questa fase, il client può inviare un **messaggio certificate_verify** per fornire verifica esplicita di un certificato client. Questo messaggio è inviato solamente in seguito a un qualunque certificato del client avente capacità di firma (per esempio, tutti i certificati ad eccezione di quelli contenenti parametri fixed Diffie-Hellman). Questo messaggio firma un codice hash sulla base dei precedenti messaggi, come segue:

```
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));
```

dove: pad_1 e pad_2 corrispondono ai valori definiti in precedenza per il MAC; handshake_messages fa riferimento a tutti i messaggi del protocollo Handshake inviati o ricevuti a partire dal messaggio client_hello escluso il messaggio corrente; master_secret è il valore segreto calcolato in base a modalità spiegate successivamente in questo

paragrafo. Se la chiave privata dell'utente è DSS, allora la si utilizza per la cifratura del valore hash SHA-1. Se la chiave privata dell'utente è RSA, allora la si utilizza per la cifratura della concatenazione dei valori hash MD5 e SHA-1. In entrambi i casi, lo scopo è quello di verificare, per il certificato del client, il possesso della chiave privata da parte del client. L'idea è che se anche qualcuno stesse facendo un cattivo uso del certificato del client, non sarebbe comunque in grado di inviare questo messaggio.

Fase 4. Conclusione

Questa fase termina l'instaurazione di una connessione sicura. Il client invia un **messaggio change_cipher_spec** e copia il valore CipherSpec dello stato in attesa nel valore CipherSpec corrente. Si noti che questo messaggio non è considerato parte del protocollo Handshake, ma viene inviato utilizzando il protocollo Change cipher spec. A questo punto il client invia immediatamente il **messaggio finished**, utilizzando i nuovi algoritmi, le chiavi e i valori segreti. Il messaggio finished verifica che i processi di scambio delle chiavi e di autenticazione abbiano avuto successo. Il contenuto del messaggio finished è la concatenazione di due valori hash:

$$\text{MD5}(\text{master\_secret} \| \text{pad2} \| \text{MD5}(\text{handshake\_messages} \| \text{Sender} \| \text{master\_secret} \| \text{pad1}))$$

$$\text{SHA}(\text{master\_secret} \| \text{pad2} \| \text{SHA}(\text{handshake\_messages} \| \text{Sender} \| \text{master\_secret} \| \text{pad1}))$$

dove Sender è un codice che identifica il fatto che il mittente è il client e handshake_messages corrisponde ai dati di tutti i messaggi di handshake fino a questo messaggio non compreso.

In risposta a questi due messaggi, il server invia il proprio messaggio change_cipher_spec, trasferisce il valore CipherSpec dello stato in attesa in quello corrente e invia il proprio messaggio finished. A questo punto, l'handshake è completo e client e server possono iniziare a scambiarsi dati a livello di applicazione.

Elaborazioni crittografiche

Altri due argomenti di particolare interesse sono la creazione di un valore master segreto condiviso mediante lo scambio di chiavi e la generazione dei parametri di crittografia a partire dal valore master segreto.

Creazione del valore master segreto

Il valore master segreto condiviso è un valore one-time di 48 byte (384 bit) generato per la sessione corrente mediante scambio sicuro di chiavi. La creazione avviene in due fasi. Prima di tutto viene scambiato un valore pre_master_secret; poi entrambe le parti calcolano il valore master_secret. Per lo scambio del valore pre_master_secret esistono due possibilità.

- **RSA.** Il client genera un valore pre_master_secret di 48 byte, cifrato con la chiave pubblica RSA del server, e lo invia al server. Il server decifra il testo cifrato ricevuto utilizzando la propria chiave privata per ricostruire il valore pre_master_secret.

- **Diffie-Hellman.** Sia il client che il server generano una chiave pubblica Diffie-Hellman. Dopo aver scambiato tali chiavi, ciascuna parte esegue il calcolo Diffie-Hellman per creare il valore condiviso pre_master_secret.

Entrambe le parti calcolano il valore master_secret come segue:

$$\begin{aligned}\text{master\_secret} = {} & \text{MD5}(\text{pre\_master\_secret} \| \text{SHA}(\text{'A'} \| \text{pre\_master\_secret} \| \\ & \quad \text{ClientHello.random} \| \text{ServerHello.random})) \| \\ & \text{MD5}(\text{pre\_master\_secret} \| \text{SHA}(\text{'BB'} \| \text{pre\_master\_secret} \| \\ & \quad \text{ClientHello.random} \| \text{ServerHello.random})) \| \\ & \text{MD5}(\text{pre\_master\_secret} \| \text{SHA}(\text{'CCC'} \| \text{pre\_master\_secret} \| \\ & \quad \text{ClientHello.random} \| \text{ServerHello.random}))\end{aligned}$$

dove ClientHello.random e ServerHello.random sono i due nonce scambiati nei messaggi hello iniziali.

Generazione dei parametri di crittografia

Le CipherSpec richiedono un valore segreto MAC di scrittura, una chiave di scrittura e un vettore IV di scrittura sia per il client che per il server, generati in tale ordine a partire dal valore master segreto. Questi parametri sono generati dal valore master segreto trasformandolo mediante un algoritmo hash in una sequenza di byte di lunghezza sufficiente a contenere tutti i parametri.

La generazione delle chiavi a partire dal valore master segreto utilizza lo stesso formato usato nella generazione del valore master segreto a partire dal valore segreto premaster:

$$\begin{aligned}\text{key\_block} = {} & \text{MD5}(\text{master\_secret} \| \text{SHA}(\text{'A'} \| \text{master\_secret} \| \\ & \quad \text{ServerHello.random} \| \text{ClientHello.random})) \| \\ & \text{MD5}(\text{master\_secret} \| \text{SHA}(\text{'BB'} \| \text{master\_secret} \| \\ & \quad \text{ServerHello.random} \| \text{ClientHello.random})) \| \\ & \text{MD5}(\text{master\_secret} \| \text{SHA}(\text{'CCC'} \| \text{master\_secret} \| \\ & \quad \text{ServerHello.random} \| \text{ClientHello.random})) \ldots\end{aligned}$$

finché viene generato un risultato di lunghezza sufficiente. Il risultato di questa struttura algoritmica è una funzione pseudocasuale. Si può considerare il valore master_secret come valore seme pseudocasuale per la funzione. I numeri casuali del client e del server possono essere considerati come valori salt per rendere più difficile la crittoanalisi (cfr Capitolo 9 per l'uso di valori salt).

TLS

TLS è un'iniziativa di standardizzazione IETF il cui obiettivo è quello di produrre una versione Internet standard di SSL. La versione corrente di Proposed standard di TLS, definita nel documento RFC 2246, è molto simile a SSLv3. In questo paragrafo vengono evidenziate le differenze.

Numero di versione

Il formato record di TLS è lo stesso di SSL (Figura 7.4) e i campi nell'intestazione hanno lo stesso significato. La sola differenza risiede nei valori del campo versione.

Nella bozza corrente di TLS, la versione principale corrisponde a 3 e la versione minore a 1.

MAC

Ci sono due differenze fra gli schemi MAC di SSLv3 e TLS: l'algoritmo utilizzato e l'ambito dell'elaborazione MAC. TLS utilizza l'algoritmo HMAC definito nel documento RFC 2104. Dal Capitolo 3, ricordiamo che HMAC è definito come segue:

$$HMAC_K = H[(K^+ \oplus opad) \| H[(K^+ \oplus ipad) \| M]]$$

dove:

H = funzione hash contenuta (per TLS, MD5 o SHA-1)
M = messaggio in ingresso a HMAC
K^+ = chiave segreta completata con una serie di zero aggiuntivi a sinistra per produrre un risultato uguale alla lunghezza del blocco utilizzato nella codifica hash (per MD5 o SHA-1, la lunghezza del blocco è 512 bit)
ipad = 00110110 (36 in esadecimale) ripetuto 64 volte (512 bit)
opad = 01011100 (5C in esadecimale) ripetuto 64 volte (512 bit)

SSLv3 usa lo stesso algoritmo ad eccezione del fatto che i byte di completamento sono concatenati con la chiave segreta piuttosto che essere combinati mediante OR esclusivo con la chiave segreta, a sua volta completata per raggiungere la necessaria lunghezza del blocco. Il livello di sicurezza dovrebbe essere lo stesso in entrambi i casi.

Per TLS, il calcolo del MAC comprende i campi riportati nella seguente espressione:

HMAC_hash(MAC_write_secret, seq_num || TLSCompressed.type ||
TLSCompressed.version || TLSCompressed.length || TLSCompressed.fragment)

Il calcolo del MAC copre tutti i campi coperti dal corrispondente calcolo di SSLv3, con in più il campo TLSCompressed.version, che indica la versione del protocollo in uso.

Funzione pseudocasuale

TLS fa uso di una funzione pseudocasuale conosciuta come PRF (*pseudorandom function*) per espandere i valori segreti in blocchi di dati per la generazione o validazione delle chiavi. L'obiettivo è utilizzare un valore segreto condiviso relativamente piccolo ma generare blocchi di dati più lunghi, in modo che l'approccio sia sicuro rispetto alle tipologie di attacchi perpetrati a funzioni hash e MAC. La funzione PRF è basata sulla seguente funzione di espansione dei dati (Figura 7.7):

P_hash(valore segreto, seme) = HMAC_hash(valore segreto, A(1) || seme)
HMAC_hash(valore segreto, A(2) || seme)
HMAC_hash(valore segreto, A(3) || seme) ...

Dove A( ) è definita come:

$$A(0) = \text{seme}$$
$$A(i) = \text{HMAC\_hash}(\text{valore segreto}, A(i-1))$$

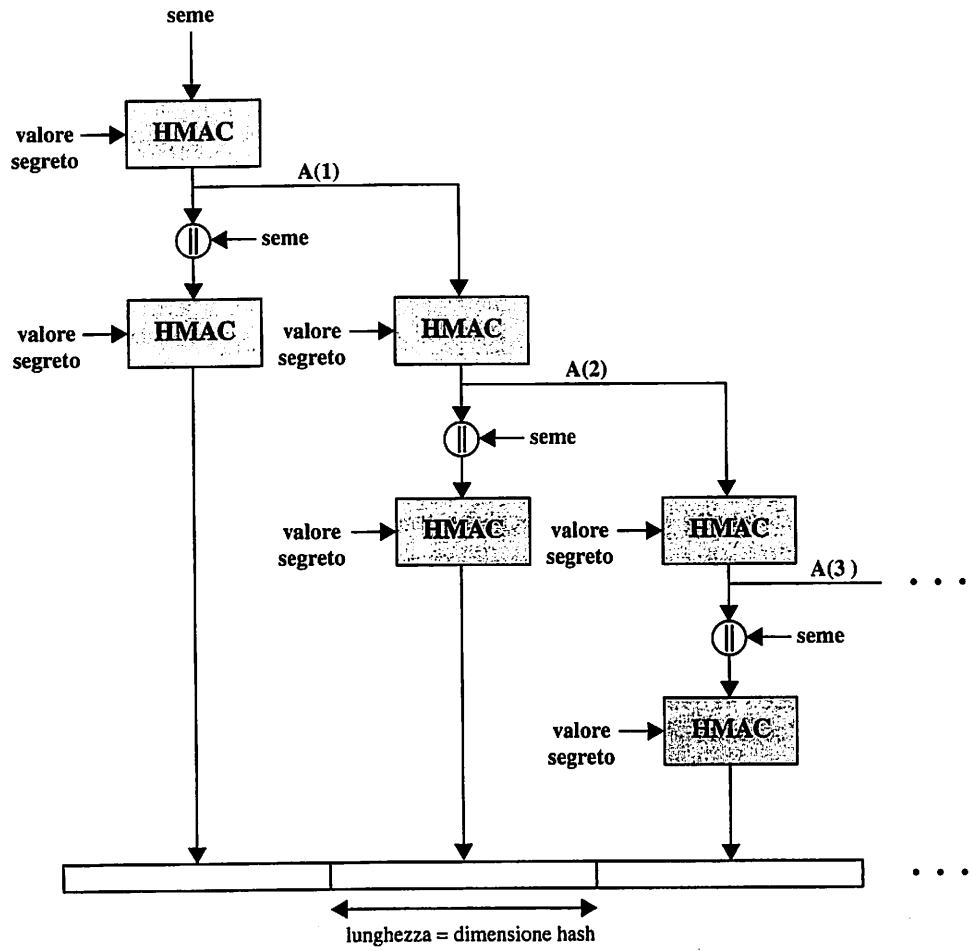

Figura 7.7 *Funzione P_hash (valore segreto, seme) di TLS.*

La funzione di espansione dei dati usa l'algoritmo HMAC, con MD5 o SHA-1 come funzioni hash sottostanti. Come si può notare, P_hash può essere iterata tante volte quante sono quelle necessarie a produrre la quantità di dati richiesta. Ad esempio, se si fosse utilizzata P_SHA-1 per generare 64 byte di dati, sarebbe stata iterata quattro volte, producendo 80 byte di dati di cui gli ultimi 16 non sarebbero stati considerati. In questo caso, anche P_MD5 sarebbe stata iterata quattro volte, producendo esattamente 64 byte di dati. Si noti che ogni iterazione richiede due esecuzioni di HMAC, ciascuna delle quali richiede a sua volta due esecuzioni degli algoritmi hash sottostanti.

Per aumentare il livello di sicurezza della funzione PRF, si usano due algoritmi hash in modo che se uno dei due algoritmi rimane sicuro, la funzione dovrebbe essere sicura. PRF è definita come:

PRF(valore segreto, etichetta, seme) = P_MD5(S1, etichetta||seme) ⊕ P_SHA-1(S2, etichetta||seme)

PRF riceve in ingresso un valore segreto, un'etichetta identificativa e un valore di seme e produce un risultato di lunghezza arbitraria. Il risultato è creato dividendo il valore segreto in parti uguali (S1 e S2) ed eseguendo la funzione P_hash su ciascuna di esse, utilizzando MD5 su una metà e SHA-1 sull'altra. I due risultati sono combinati mediante OR esclusivo per produrre il risultato finale; a tale scopo, P_MD5 deve essere iterata generalmente più volte di P_SHA-1 per produrre una pari quantità di dati in ingresso alla funzione di OR esclusivo.

Codici di allarme

TLS supporta tutti i codici di allarme definiti in SSLv3 ad eccezione del codice no_certificate. In TLS sono definiti un certo numero di codici aggiuntivi; fra questi, i seguenti denotano situazioni sempre fatali.

- **decryption_failed:** decifratura non valida di un testo cifrato; o non era un multiplo pari della lunghezza del blocco oppure i valori di completamento, in fase di verifica, risultavano errati.
- **record_overflow:** è stato ricevuto un record TLS con un payload (testo cifrato) avente lunghezza superiore a 2^{14} + 2048 byte, oppure il testo cifrato è stato decifrato a una lunghezza superiore a 2^{14} + 1024 byte.
- **unknown_ca:** è stata ricevuta una catena di certificati valida o una catena parziale, ma il certificato non è stato accettato in quanto il certificato della CA non può essere localizzato o non può essere confrontato con una CA conosciuta e fidata.
- **access_denied:** è stato ricevuto un certificato valido, ma nell'applicare il controllo dell'accesso, il mittente ha deciso di non procedere con la negoziazione.
- **decode_error:** non è possibile decodificare un messaggio in quanto un campo contiene un valore al di fuori dell'intervallo di valori ammissibili specificato, oppure la lunghezza del messaggio era errata.
- **export_restriction:** è stata scoperta una negoziazione non ottemperante alle restrizioni di esportazione circa la lunghezza della chiave.
- **protocol_version:** la versione di protocollo che il client ha tentato di negoziare è riconosciuta ma non supportata.
- **insufficient_security:** valore ritornato al posto di handshake_failure quando una negoziazione è fallita per il fatto che il server richiede cifrari più sicuri di quelli supportati dal client.
- **internal_error:** un errore interno non connesso alla parte o alla correttezza del protocollo rende impossibile la continuazione.

I nuovi allarmi rimanenti sono i seguenti.

- **decrypt_error:** un'operazione di handshake di cifratura è fallita, comprendendo i casi di incapacità di verificare una firma, di decifrare uno scambio di chiavi o di validare un messaggio finished.

- **user_canceled:** l'operazione di handshake corrente viene cancellata per ragioni non connesse a un malfunzionamento del protocollo.
- **no_renegotiation:** inviato da un client in risposta a una richiesta hello oppure dal server in risposta a un messaggio hello del client dopo l'handshake iniziale. Un messaggio qualunque di questi porta normalmente alla rinegoziazione, ma questo allarme indica che il mittente non è in grado di effettuarla. Questo messaggio è sempre un messaggio di avviso.

Suite di cifratura

Esistono numerose, piccole differenze fra le suite di cifratura disponibili in SSLv3 e TLS.

- **Scambio di chiavi:** TLS supporta tutte le tecniche di scambio di chiavi di SSLv3 ad eccezione di Fortezza.
- **Algoritmi di cifratura simmetrica:** TLS comprende tutti gli algoritmi di cifratura simmetrica di SSLv3, ad eccezione di Fortezza.

Tipologie di certificato del client

TLS definisce le seguenti tipologie di certificati da richiedere in un messaggio certificate_request: rsa_sign, dss_sign, rsa_fixed_dh, e dss_fixed_dh. Tutti questi sono definiti in SSLv3. Inoltre, SSLv3 comprende rsa_ephemeral_dh, dss_ephemeral_dh, e fortezza_kea. Ephemeral Diffie-Hellman richiede la firma dei parametri Diffie-Hellman con RSA o DSS; per TLS, i tipi usati sono rsa_sign e dss_sign. Non è necessario un tipo separato di firma per firmare i parametri Diffie-Hellman. TLS non comprende lo schema Fortezza.

Messaggi certificate_verify e finished

Nel messaggio certificate_verify di TLS, i codici hash MD5 e SHA-1 sono calcolati solamente su handshake_messages. Nel caso di SSLv3, il calcolo del valore hash comprende anche il valore master segreto e i pad. L'impressione era che questi campi addizionali non aggiungessero sicurezza ulteriore.

Come per il messaggio finished in SSLv3, anche in TLS questo messaggio è un codice hash basato sul valore condiviso master_secret, sui messaggi handshake precedenti e su un'etichetta che identifica il client o il server. Il calcolo differisce leggermente. Nel caso di TLS, si ha:

PRF(master_secret, finished_label, MD5(handshake_messages)||SHA-1 (handshake_messages))

dove finished_label è la stringa "client finished" per il client e "server finished" per il server.

Elaborazioni di crittografia

Il valore pre_master_secret di TLS viene calcolato allo stesso modo del corrispondente valore di SSLv3. Come in SSLv3, il valore master_secret di TLS è calcolato come

una funzione hash del valore pre_master_secret e dei due numeri casuali di hello. L'elaborazione in TLS differisce da quella di SSLv3 ed è definita come segue:

```
master_secret =
  PRF(pre_master_secret, "master secret", ClientHello.random || ServerHello.random)
```

L'algoritmo viene eseguito fino a produrre un risultato pseudocasuale di 48 byte. Il calcolo del blocco di informazioni relative alle chiavi (chiavi segrete MAC, chiavi di cifratura di sessione e IV) è definito come segue:

```
key_block =
  PRF(master_secret, "key expansion",
    SecurityParameters.server_random || SecurityParameters.client_random)
```

finché non viene generato un risultato di dimensione sufficiente. Come per SSLv3, key_block è una funzione del valore master_secret e dei numeri casuali del client e del server, ma l'algoritmo di TLS è diverso.

Completamento

In SSL, il completamento aggiunto prima di effettuare la cifratura dei dati utente corrisponde alla quantità minima necessaria a rendere la dimensione totale dei dati da cifrare multipla della lunghezza del blocco del cifrario. In TLS, il completamento può essere una quantità qualunque che porti ad avere un risultato multiplo della lunghezza del blocco del cifrario, fino a un massimo di 255 byte. Ad esempio, se il testo in chiaro (o quello compresso se si usa la compressione), insieme con il MAC e il byte padding.length hanno una lunghezza totale di 79 byte, allora la lunghezza in byte del completamento può essere 1, 9, 17, e così via, fino a 249. Una lunghezza variabile per il completamento può essere utilizzata per scoraggiare attacchi basati sull'analisi della lunghezza dei messaggi scambiati.

7.3 Secure electronic transaction (SET)

SET è una specifica di cifratura e sicurezza progettata per fornire protezione alle transazioni effettuate con carta di credito su Internet. La versione corrente, SETv1, è il frutto di una richiesta di standard di sicurezza avanzata da MasterCard e Visa nel febbraio 1996. Nello sviluppo della specifica iniziale erano coinvolte numerose aziende fra cui IBM, Microsoft, Netscape, RSA, Terisa e VeriSign. A partire dal 1996, sono state effettuate numerose verifiche della specifica iniziale e nel 1998 era disponibile la prima serie di prodotti rispondenti alle specifiche SET.

SET non è di per sé un sistema di pagamento; consiste piuttosto in un insieme di protocolli e formati di sicurezza che consentono agli utenti di utilizzare, in maniera sicura, l'infrastruttura di pagamento basata su carta di credito su una rete aperta, come ad esempio Internet. Essenzialmente, SET fornisce tre servizi:

- un canale di comunicazione sicuro fra le parti coinvolte in una transazione;
- affidabilità attraverso l'uso di certificati digitali X.509v3;