Elementi di Sicurezza informatica

Secure Socket Layer

Federico Reali

Università degli Studi di Perugia

Indice

Introduzione

Introduzione

- Perché SSL;
- Funzionalità;
- La storia di SSL.

Perché SSL

TCP

Il protocollo TCP fornisce un affidabile e efficiente canale di comunicazione tra due nodi e permette di:

- Individuare la perdita di pacchetti;
- Individuare pacchetti corrotti;
- Non tiene conto di dati trasmessi erroneamente più volte.

Tuttavia..

- Non garantisce l'autenticazione (è basato sull'autenticazione per indirizzo);
- Non fornisce garanzie di confidenzialità (i dati passano in chiaro sul canale);
- Non fornisce garanzie di integrità.

Perché SSL

Soluzione..

Secure Socket Layer (SSL) nasce proprio per rimediare alle mancanze di TCP.
In particolare SSL è un protocollo di due layer che si vanno ad inserire nel Internet Protocol, tra il livello di trasporto e il livello delle applicazioni.
Questa specifica ubicazione permette a SSL di ereditare le proprietà di TCP, senza dover trattare nello specifico la gestione del trasporto dei dati.

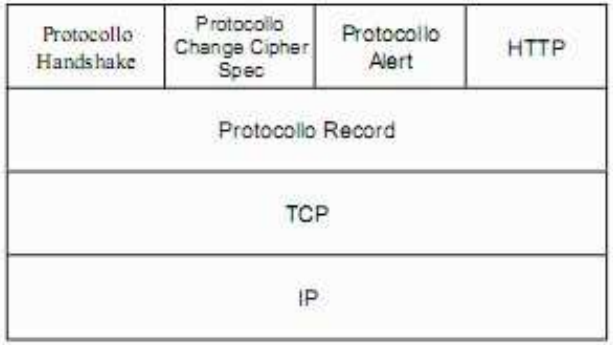


Figura: Stack del protocollo SSL

Cosa Assicura

Confidenzialità

I dati prima di essere inviati nel canale di comunicazione vengono cifrati attraverso algoritmi di crittografia a chiave simmetrica.

Autenticazione

Assicura l'identità dei soggetti o attraverso la crittografia a chiave pubblica, o attraverso l'uso di certificati.

Integrità

Il protocollo include un controllo di integrità sul messaggio basato su MAC, utilizzando funzioni hash sicure come SHA e MD-5.

Cosa Assicura

Negoziazione

Permette ad un server ed un client di negoziare gli algoritmi di cifratura e le funzioni hash che verranno utilizzati in base a quelli 'noti' a ciascuno dei due.

Riutilizzo

Permette di riutilizzare i parametri negoziati in precedenza, velocizzando il funzionamento dello stesso.

Storia di SSL

La Genesi

SSL fu creato da Netscape Communications Corporation con l'intento di assicurare gli obiettivi di sicurezza visti in precedenza.

Versioni

- 1994 Prima versione di SSL. Essa presentava diversi problemi e non venne mai utilizzata;
- 1994 Seconda versione implementata in Netscape Navigator 1;
- 1996 Terza versione implementata in Netscape Navigator 3;
- 1999 TLS (Transport Layer Security) RFC2246 evoluzione di SSL proposta da IETF;
- 2002 TLS RFC3268 Ulteriore evoluzione che supporta anche AES e IDEA.

Storia di SSL

Figura: Screenshot di Netscape Navigator con la possibilità di scegliere tra SSL v2 o SSL v3.

Utilizzo

e-Commerce

SSL/TLS è utilizzato nel commercio elettronico per ordini o per pagamenti.

Figura: Login su Amazon.co.uk con Google Chrome

Figura: Ordine su Amazon.co.uk con Firefox

Utilizzo

Accesso 'Sicuro'

SSL/TLS è utilizzato per l'invio di password o altri dati riservati e per la consultazione di informazioni con restrizioni di accesso.

Figura: Login su Ebay con Opera

Figura: Login su SOL unipg con Firefox

Utilizzo

HTTPS e oltre

Come si è potuto notare dalle immagini precedenti, TLS è un protocollo implementato da tutti i maggiori browser grazie ad HTTPS (Hypertext Transfer Protocol over Secure Socket Layer) che integra HTTP con SSL/TLS . Esso inoltre è applicabile anche ad altri protocolli come POP3, IMAP e FTP. Esiste anche una implementazione open source disponibile per i più diffusi sistemi operativi chiamata **OpenSSL** la cui versione 1.0.0 è stata rilasciata il 29 Marzo 2010.

Utilizzo

Date	Newsflash
29-Mar-2010:	OpenSSL 1.0.0 is now available, a major release
24-Mar-2010:	Security Advisory: "Record of death" security fix
24-Mar-2010:	OpenSSL 0.9.8n is now available, including important bug and security fixes
25-Feb-2010:	OpenSSL 0.9.8m is now available, including important bug and security fixes
24-Jun-2009:	Commercial support for OpenSSL is now available

more...

Figura: OpenSSL disponibile su `http://www.openssl.org/`

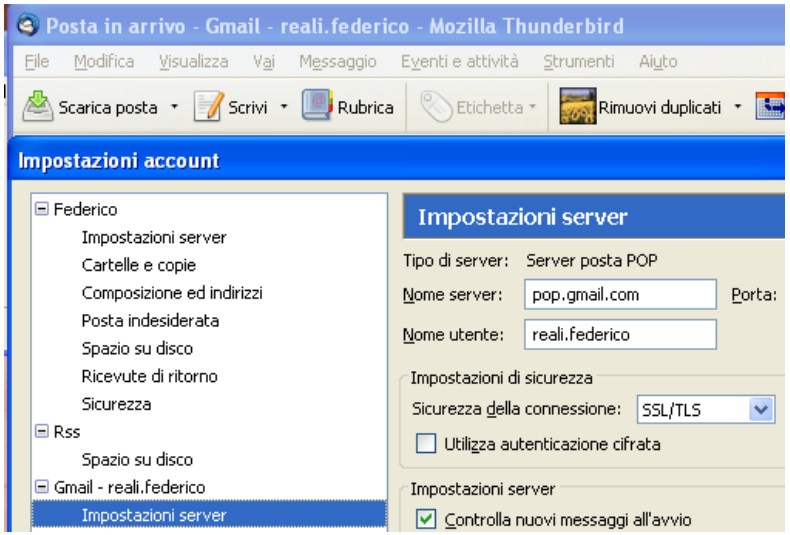


Figura: Opzioni per SSL su Thunderbird

Il Protocollo

SSL

Come è già stato detto in precedenta SSL incorpora in se due layer: uno a livello più basso e uno a livello più alto. Quest'ultimo è a sua volta composto da più protocolli.
Nelle slides che seguono verranno analizzati nel dettaglio i due layer e i protocolli che li compongono.

Il Protocollo

Di particolare importanza per la comprensione del protocollo risultano queste due definizioni:

Connessione

E' una connessione a livello trasporto che fornisce un certo tipo di servizi. Per SSL tali connessioni sono relazioni *peer-to-peer*. Ogni connessione è associata ad una sessione.

Sessione

Una sessione SSL è un associazione tra un client e un server. Le sessioni sono create attraverso il protocollo Handshake. Le sessioni definiscono una serie di parametri di sicurezza, che possono essere condivisi tra più connessioni. Le sessioni servono per evitare costose negoziazioni di nuovi parametri di sicurezza per ogni connessione.

Algoritmi per lo scambio delle chiavi

E questi algoritmi per lo scambio delle chiavi:

- RSA: Chiavi segrete cifrate con la chiave pubblica del ricevente, autenticate tramite certificato;
- Fixed Diffie-Hellman: Algoritmo Diffie-Hellman in cui il certificato del server contiene i parametri pubblic Diffie-Hellman forniti da una CA (Certification Autority). Il client fornisce i propri parametri in un certificato oppure in un messaggio durante lo scambio di chiavi;
- Ephemeral Diffie-Hellman: Viene utilizzato per creare chiavi segrete temporanee usando parametri pubblici *one-time*, scambiati e firmati usando RSA o simili. E' il metodo ritenuto più sicuro.
- Anonymus Diffie-Hellman: Come Diffie-Hellman classico ma entrambe le parti spediscono i propri parametri pubblici senza autenticazione.

SSL Record Protocol

SSL Record Protocol

E' il protocollo che materialmente prende i dati ai livelli superiori e li sottopone al trattamento di sicurezza, prima di passarli al livello sottostante.
Esso fornisce:

Confidenzialità Si usano la chiave segreta e gli algoritmi di encryption concordati;

Integrità del Messaggio Attraverso l'uso di MAC.

La figura che segue riporta nel dettaglio come opera tale protocollo.

SSL Record Protocol

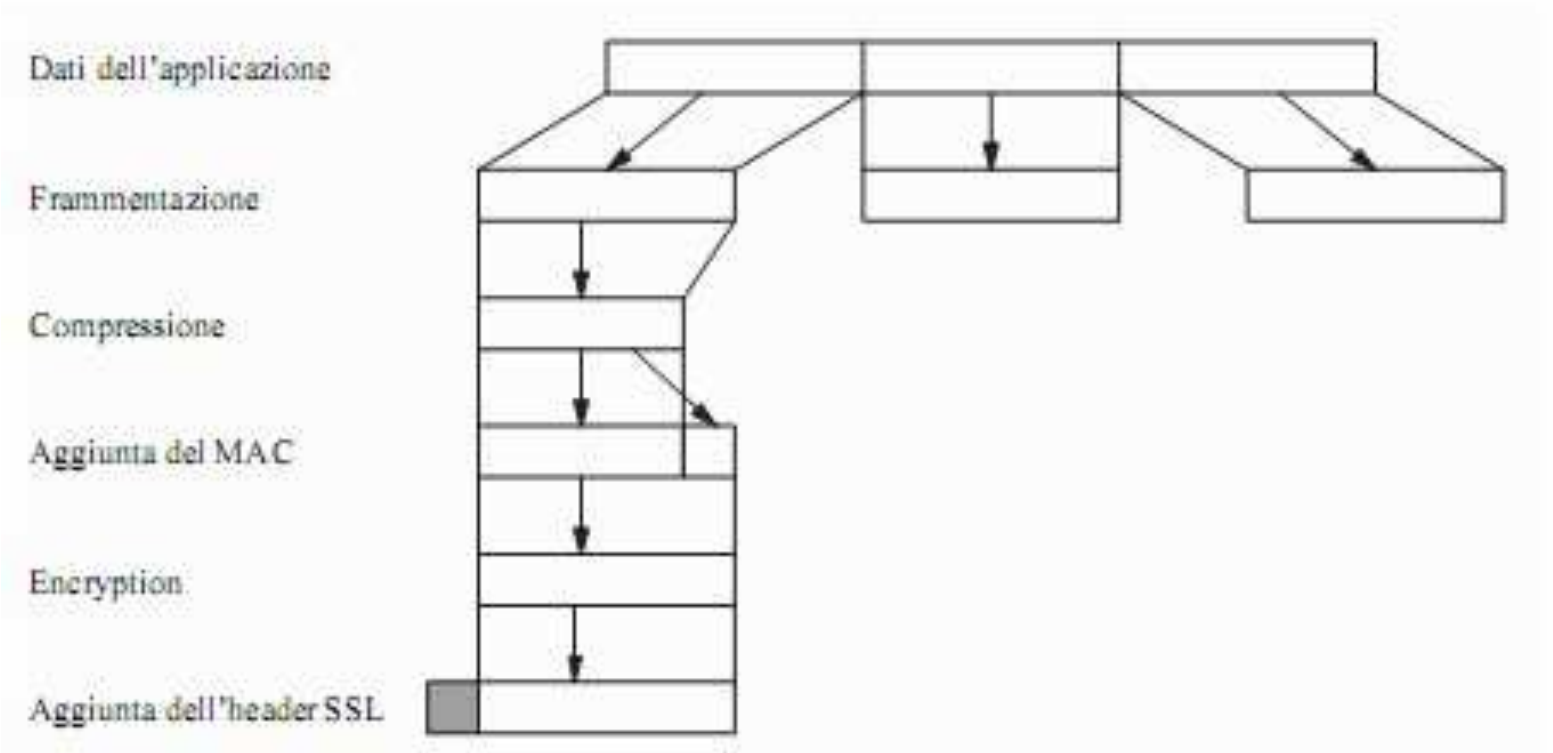


Figura: Operazioni del Record Protocol

SSL Record Protocol

Come Lavora

I dati dai layer superiori del protocollo vengono prima frammentati in blocchi, poi compressi (se specificato), dopodiché viene calcolato il MAC sui dati compressi. A questo punto il messaggio compresso più il MAC vengono cifrati, e per ultimo viene apposto l'header del protocollo SSL Record.
Il campo *Content Type* indica il tipo di dati contenuti nel frammento blocco, ovvero il protocollo che verrà usato per elaborare tali dati; il campo *Compressed Length* contiene la lunghezza del blocco di plaitext (compresso o meno) contenuto nel messaggio.

SSL Record Protocol

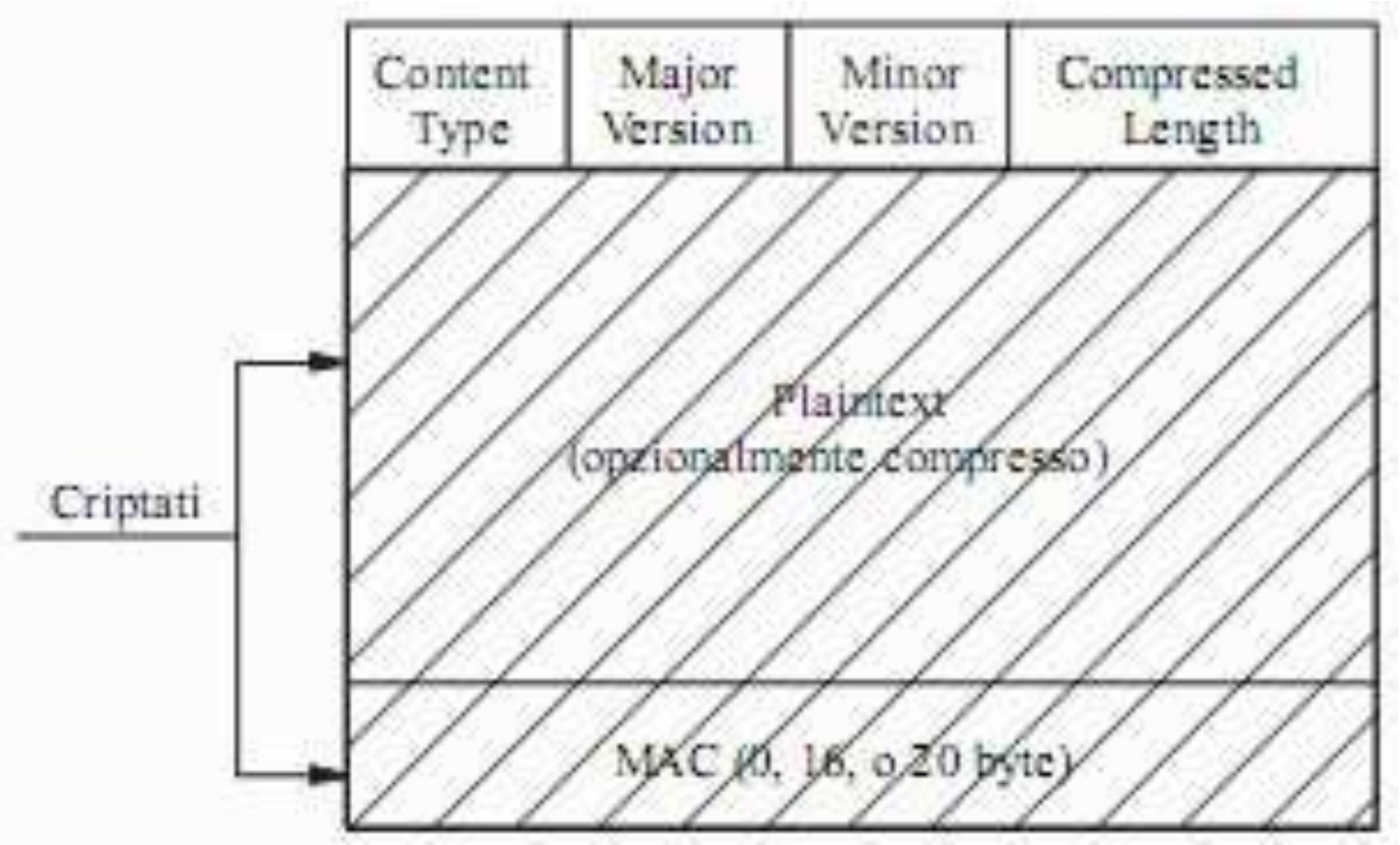


Figura: Operazioni del Record Protocol

Alert Protocol

Alert Protocol

Il protocollo alert è usato per trasportare messaggi di allarme, compressi e cifrati.
Ogni messaggio di questo protocollo consiste di due byte.
Il primo byte rappresenta la gravità dell'allarme spedito (*warning* oppure *fatal*).
Il secondo byte contiene un codice che indica la motivazione del messaggio.
Nel caso in cui il livello di gravità sia fatal, la connessione SSL viene terminata immediatamente.

Change Cipher Spec Protocol

Change Cipher Spec Protocol

Il protocollo Change Cipher Spec è il più semplice dei protocolli utilizzati da SSL.
Esso consiste in un singolo messaggio, costituito da un solo byte con valore 1. L'unica funzione di questo messaggio è fare in modo che siano aggiornate le informazioni riguardanti la suite di algoritmi di cifratura da utilizzare nella connessione, nel caso questi siano stati rinegoziati a livello di sessione tramite Handshake.

Handshake Protocol

Handshake Protocol

E' di gran lunga il più complesso e permette al server e al client di:

- Autenticarsi a vicenda, negoziando il metodo di autenticazione;
- Negoziare un algoritmo di encryption;
- Negoziare un algoritmo per calcolare il MAC;
- Stabilire le chiavi segrete usate per cifrare i dati contenuti nei record SSL, negoziando il metodo per farlo.

Il protocollo Handshake viene usato prima della trasmissione dei dati delle applicazioni.
Nelle slides seguenti analizzeremo nel dettaglio i possibili messaggi scambiati durante tale protocollo.

Handshake Protocol

Prima Fase

In essa vengono definite le specifiche per la sicurezza.
Questa fase è utilizzata per iniziare una connessione logica e stabilire le specifiche di sicurezza che le saranno associate.
Lo scambio è iniziato dal client, che spedisce un messaggio *client-hello* con cui informa il server sugli algoritmi di cifratura e di compressione supportati, e spedisce un identificatore di sessione:
un valore diverso da zero di tale parametro indica la volontà del client di aggiornare i parametri di una connessione esistente o di avviarne una nuova, mentre un valore uguale a zero invece indica la volontà di stabilire una nuova connessione in una nuova sessione.
Dopodiché attende dal server un messaggio *server-hello*, del tutto simile a quello spedito dal client. L'unica particolarità sta nell'identificatore di sessione: se quello del client era diverso da zero lo stesso sarà per questo identificatore, altrimenti esso conterrà il nuovo valore generato dal server per questa sessione;

Handshake Protocol

Seconda Fase

In essa avviene l'autenticazione del server e lo scambio delle chiavi. E' una fase in cui solo il server spedisce messaggi. Inizia con un messaggio *certificate*, necessario solo se si utilizza un metodo di negoziazione della chiave diverso da Anonymous Diffie-Hellman. A questo punto segue un messaggio *server-key-exchange*, che non è richiesto solo nel caso in cui sia stato mandato un messaggio certificate con i parametri Fixed Diffie-Hellman oppure venga usato RSA per lo scambio di chiavi. La firma contenuta in questo messaggio è ottenuta calcolando un valore hash sui parametri di Diffie-Hellman o RSA e sui nonce presenti nei messaggi *hello* iniziali (per evitare attacchi di tipo replay), e cifrando tale valore con la chiave privata del server. Quello che segue è un messaggio di *certificate-request*, per richiedere l'autenticazione del client e può essere spedito solo da un server che non usi Anonymous Diffie-Hellman. Infine viene spedito un messaggio *server-hello-done*, che chiude la seconda fase.

Handshake Protocol

Terza Fase

In questa fase avviene l'autenticazione del client e lo scambio delle chiavi. In questa fase il client procede alla verifica della validità del certificato del server e dei parametri del messaggio *server-hello*. Se tutto va bene il client spedisce un messaggio *certificate*, nel caso sia stato richiesto. Dopodiché spedisce un *client-key-exchange*, per lo scambio della chiave: esso contiene i parametri necessari, a seconda del tipo di algoritmo utilizzato. Vediamo alcuni esempi:

- Se l'algoritmo usato è RSA il client genera un *pre-master secret* e lo cifra con la chiave pubblica del certificato del server o con la chiave temporanea RSA appartenente al messaggio *server-key-exchange*. Il suo utilizzo verrà trattato in seguito;
- Se l'algoritmo è Ephemeral o Anonymous Diffie-Hellman, vengono spediti i parametri necessari;
- Se l'algoritmo è Fixed Diffie-Hellman, i parametri necessari vengono spediti in un messaggio *certificate*, quindi questo messaggio è vuoto.

Handshake Protocol

Terza Fase Continua

Infine il client può trasmettere un messaggio *certificate-verify* per fornire la verifica esplicita del proprio certificato. Questo messaggio contiene una firma, costituita dalla cifratura di un valore hash calcolato sui precedenti messaggi, con la propria chiave privata, in modo che sia possibile verificare che la chiave privata appartiene al certificato del client. Nel calcolare il valore hash viene utilizzato un valore chiamato *master secret*.

Master Secret

Il master secret condiviso è un valore temporaneo a 48 bit calcolato durante lo scambio delle chiavi. La sua creazione avviene in due passi. Prima viene scambiato un *pre-master secret* (Fase 3). Dopodiché entrambe le parti ottengono il master secret concatenando tre valori hash calcolati sul pre-master secret e sui due nonce scambiati inizialmente.

Handshake Protocol

Quarta Fase

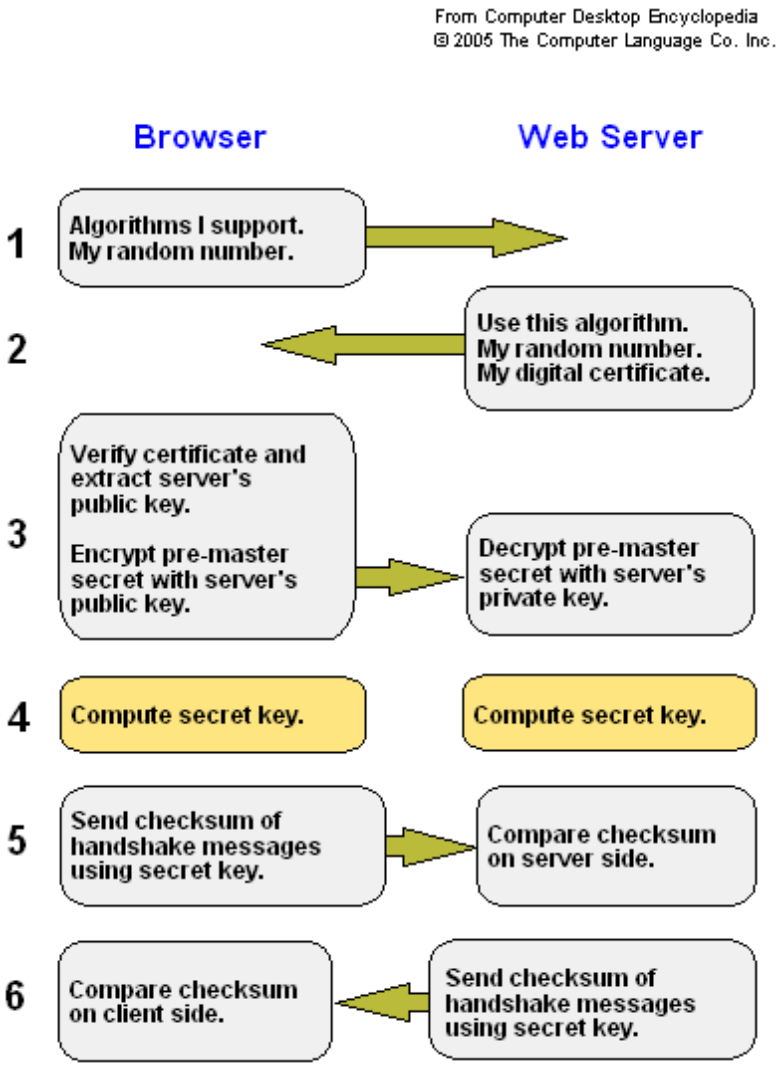
Questa fase completa la creazione di una connessione sicura.
Il client spedisce un messaggio *change-Cipher-spec*, agganciando le specifiche riguardo ai metodi di encryption usati (*CipherSpec*), contenute anche nella *Suite di cifrature* del messaggio *client-hello* (*server-hello*), secondo quanto concordato in precedenza. Dopodiché manda un messaggio *finished* per terminare la comunicazione.
Tale messaggio contiene la concatenazione di due valori hash, calcolati sul master secret e sui messaggi precedenti, uno con SHA-1 e l'altro con MD-5. Il server, come risposta, fa le stesse cose e termina lo scambio di messaggi. A questo punto la connessione è stabilita.

Graficamente

Protocollo

La figura nella slide che segue rappresenta a grandi linee le azioni del protocollo SSL Handshake, mentre le slides successive analizzano nel dettaglio un possibile scambio di messaggi dello stesso protocollo.

HandShake


From Computer Desktop Encyclopedia
© 2005 The Computer Language Co. Inc.
Browser
Web Server
1
Algorithms I support.
My random number.
2
Use this algorithm.
My random number.
My digital certificate.
3
Verify certificate and extract server's public key.
Encrypt pre-master secret with server's public key.
Decrypt pre-master secret with server's private key.
4
Compute secret key.
Compute secret key.
5
Send checksum of handshake messages using secret key.
Compare checksum on server side.
6
Compare checksum on client side.
Send checksum of handshake messages using secret key.

Hello-Client

M1: ClientHello:

ClientRandom[28]
Suggested Cipher Suites: TLS_RSA_WITH_IDEA_CBC_SHA TLS_RSA_WITH_3DES_EDE_CBC_SHA TLS_DH_DSS_WITH_AES_128_CBC_SHA
Suggested Compression Algorithm: NONE

Server-Hello, Certificate, Server-Done

M2:

ServerHello:
ServerRandom[28]
Use Cipher Suite: TLS_RSA_WITH_3DES_EDE_CBC_SHA
Session ID: 0xa00372d4XS

Certificates:
subjectAltName: SuperStoreVirtualOutlet PublicKey: 0x521aa593 ... Issuer: SuperStoreHQ
subjectAltName: SuperStoreHQ PublicKey: 0x9f400682 ... Issuer: Verisign

Server Done:
NONE

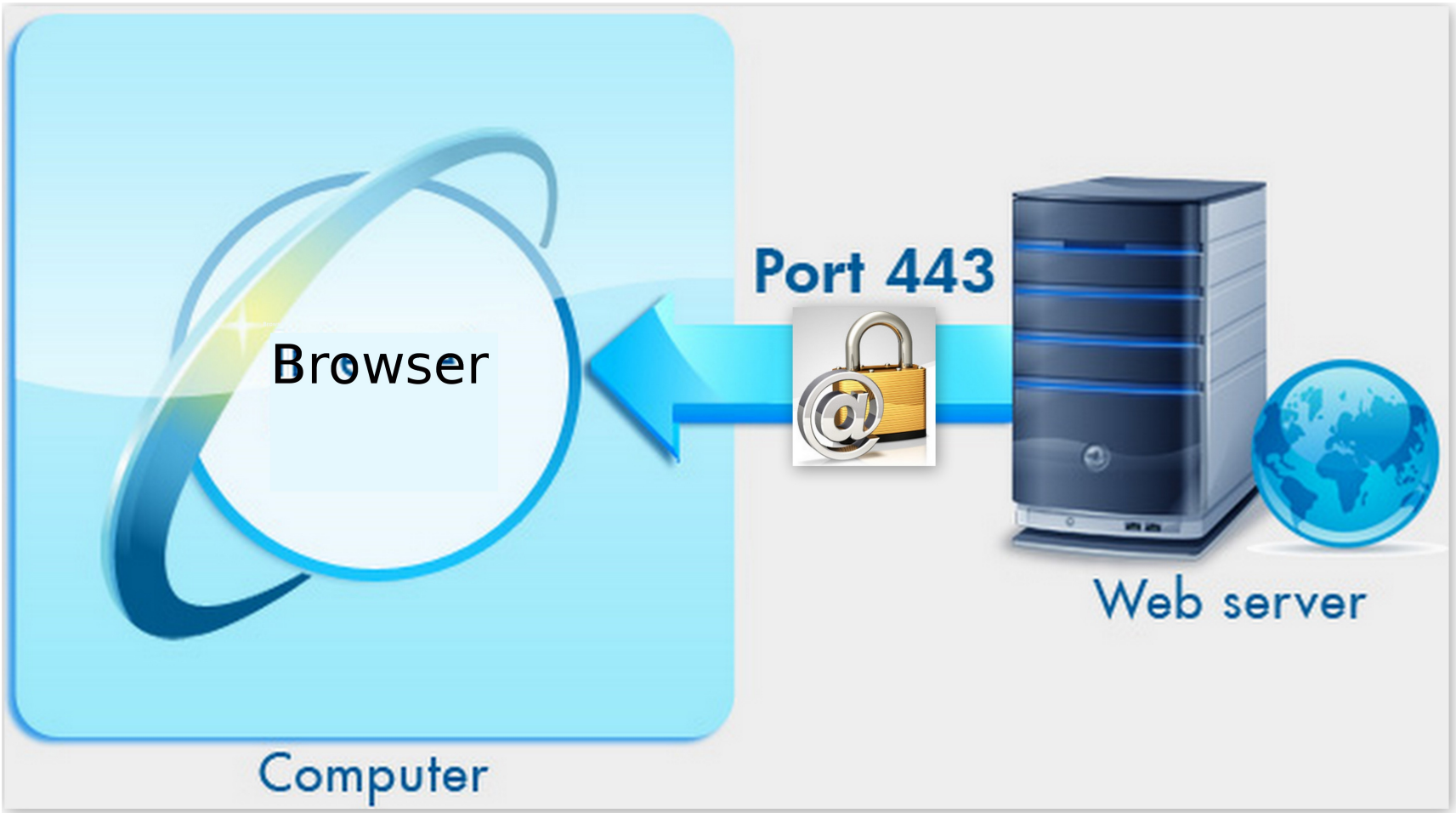
Client-Key-Exchange, ChangeCipherSpec, Finished

M3:	A: ClientKeyExchange:	RSA_Encrypt(ServerPublicKey,PreMasterSecret)
	B: ChangeCipherSpec:	NONE
	C: Finished	MD5(M1 || M2 || M3A) SHA(M1 || M2 || M3A)

Server-ChangeCipherSpec, Finished

M4:	A: ChangeCipherSpec:	NONE
	B: Finished	MD5(M1 || M2 || M3A || M3C) SHA(M1 || M2 || M3A || M3C)

Sicuri!



Grazie!