Secure Crypto
Protocollo di sicurezza basato su Chiave Pubblica e segreto condiviso tra Client e Server

Carmine Benedetto

25 Gennaio 2012

Indice

1 Specifiche Formali

1.1 Indicazioni generali

Si consideri un'applicazione distribuita di tipo cliente-servitore in cui il server ha una coppia di chiavi pubblica e privata e la chiave pubblica è nota ai clienti. Ciascun cliente condivide una password segreta con il server.

1.2 Requisiti

Si specifichi, si analizzi, si progetti ed, infine, si implementi un protocollo crittografico che soddisfi i seguenti requisiti:

- al termine dell'esecuzione del protocollo, viene stabilita una chiave di sessione tra cliente e servitore;
- al termine dell'esecuzione del protocollo, il cliente ritiene che il servitore dispone della chiave di sessione e viceversa.

1.3 Richieste avanzate

La specifica del protocollo deve mettere chiaramente in evidenza le ipotesi sotto le quali il protocollo funziona correttamente.
La specifica del protocollo deve comprendere la realizzazione di un prototipo in cui il server ed il cliente si scambiano del materiale (testo o binario) cifrato con la chiave di sessione.

2 Analisi del protocollo

2.1 Descrizione

Il protocollo complessivamente consta di quattro messaggi.

1. Il messaggio M1 viene inviato in chiaro dal Client al Server e trasporta gli identificativi del Client e del Server. Tramite il suddetto messaggio il Client comunica al Server l'intenzione di instaurare una connessione con lo scopo di stabilire una chiave di sessione.

2. Il Server risponde al Client con il messaggio in chiaro M2 che contiene gli identificativi di Server e Client accompagnati da un nonce.

3. Il Client invia al Server il messaggio M3, cifrato con la chiave pubblica del Server, contenente i soliti identificativi, il nonce precedentemente inviato dal Server, la password condivisa tra Client e Server e la chiave di sessione generata dal Client.

4. Il Server conclude il protocollo inviando al Client il messaggio M4, cifrato con la chiave di sessione, in cui inoltra i soliti identificativi e la password condivisa.

2.2 Visione formale

M1: C $\rightarrow$ S: C, S

M2: S $\rightarrow$ C: S, C, N

M3: C $\rightarrow$ S: E_{Kpub}(C, S, N, P, Kses)

M4: S $\rightarrow$ C: E_{Kses}(S, C, P)

2.3 Visione idealizzata

I messaggi non cifrati, non avendo rilevanza ai fini dell'analisi, non verrano presi in considerazione.

M3: C $\rightarrow$ S: $\{\langle \text{N, C} \Longleftrightarrow^{P} \text{S, C} \longleftrightarrow^{Kses} \text{S}\rangle_{P}\}_{Kpub}$

M4: S $\rightarrow$ C: $\{\langle \text{S} \Longleftrightarrow^{P} \text{C, C} \longleftrightarrow^{Kses} \text{S}\rangle_{P}\}_{Kses}$

2.4 Note

La chiave di sessione viene generata dal Client in maniera random quindi ha una validità temporale limitata ed è considerabile fresca.
Il Server ritiene che il Client sia un'entità affidabile per la generazione della chiave di sessione.
Il Client ha a disposizione la chiave pubblica del Server che si suppone essere certificata da un apposito ente.
Il nonce viene generato dal Server in maniera random quindi ha una validità temporale limitata ed è considerabile come una quantità fresca.
Nel messaggio M4, la password condivisa viene inserita all'interno del messaggio per avere ulteriore conferma che gli estremi della comunicazione non siano cambiati e per incrementare il corpo del messaggio in modo da rendere più complicati attacchi di tipo esaustivo.

2.5 Ipotesi iniziali

Osservando il protocollo possiamo dedurre semplicemente delle ipotesi iniziali:

$\mathrm{S} \mid\equiv \sharp(\mathrm{N})$

$\mathrm{S} \mid\equiv \mathrm{C} \Longleftrightarrow^{P} \mathrm{S}$

$\mathrm{C} \mid\equiv \mathrm{C} \Longleftrightarrow^{P} \mathrm{S}$

$\mathrm{C} \mid\equiv \hookrightarrow^{Kpub} S$

Dallo studio dei messaggi M3 ed M4 risulta ovvio che:

$\mathrm{S} \lhd \{\mathrm{N},\ \mathrm{C} \Longleftrightarrow^{P} \mathrm{S},\ \mathrm{C} \longleftrightarrow^{Kses} \mathrm{S}\}_{Kpub}$

$\mathrm{C} \lhd \{\mathrm{C} \Longleftrightarrow^{P} \mathrm{S},\ \mathrm{C} \longleftrightarrow^{Kses} \mathrm{S}\}_{Kses}$

2.6 Produzione delle asserzioni e aggiunta delle ipotesi

Nel messaggio M3 per la *message meaning rule* otteniamo:

$\mathrm{S} \mid\equiv \mathrm{C} \mid\sim (\mathrm{N},\ \mathrm{P},\ \mathrm{C} \longleftrightarrow^{Kses} \mathrm{S})$

Per la *nonce verification rule* otteniamo:

S $|\equiv$ C $|\equiv$ (N, P, C $\longleftrightarrow^{Kses}$ S)

Aggiungendo le ipotesi:

S $|\equiv$ C $\Rightarrow$ C $\longleftrightarrow^{Kses}$ S

S $|\equiv$ C $\Rightarrow \sharp$(C $\longleftrightarrow^{Kses}$ S)

otteniamo tramite la *jurisdition rule*:

S $|\equiv$ C $\longleftrightarrow^{Kses}$ S

S $|\equiv \sharp$ (C $\longleftrightarrow^{Kses}$ S)

Risulta facilmente deducibile che valgono anche:

C $|\equiv$ C $\longleftrightarrow^{Kses}$ S

C $|\equiv \sharp$ (C $\longleftrightarrow^{Kses}$ S)

Nel messaggio M4, utilizzando la *message meaning rule* otteniamo:

C $|\equiv$ S $|\sim$ (C $\Longleftrightarrow^{P}$ S, C $\longleftrightarrow^{Kses}$ S)

Utilizzando l'asserzione sulla freschezza della chiave prodotta precedentemente otteniamo:

C $|\equiv$ S $|\equiv$ (C $\Longleftrightarrow^{P}$ S, C $\longleftrightarrow^{Kses}$ S)

2.7 Verifica per deduzione logica

Il Client, avendo ricevuto da un ente di certificazione la chiave pubblica del Server, la utilizza per cifrare il messaggio contenente la chiave di sessione da esso generata. In questo modo è sicuro che solo il Server, attraverso la chiave privata, sarà in grado di decifrare il messaggio e quindi di venire in possesso della chiave di sessione. Permane il fatto che il Server pone fiducia nella capacità del Client di generare in maniera adeguata e affidabile la chiave di sessione.
Il Server, al fine di confermare al Client di aver ricevuto in maniera corretta la chiave di sessione, invia un messaggio cifrato con la stessa, contenente

parte del messaggo precedentemente ricevuto. La presenza nel messaggio del segreto condiviso tra Client e Server, da conferma che il messaggio proviene dal Server.

2.8 Verifica degli obiettivi

Raccogliendo le asserzioni prodotte e facendo delle semplici deduzioni basate sull'osservazione del protocollo notiamo che risultano rispettati gli obiettivi di:

- **key authentication**: C $|\equiv$ C $\longleftrightarrow^{Kses}$ S, S $|\equiv$ C $\longleftrightarrow^{Kses}$ S
- **key confirmation**: C $|\equiv$ S $|\equiv$ C $\longleftrightarrow^{Kses}$ S, S $|\equiv$ C $|\equiv$ C $\longleftrightarrow^{Kses}$ S
- **key freshness**: C $|\equiv \sharp$ (C $\longleftrightarrow^{Kses}$ S), S $|\equiv \sharp$ (C $\longleftrightarrow^{Kses}$ S)

3 Implementazione del protocollo

3.1 Ambiente di sviluppo e linguaggio di programmazione

Il progetto è stato realizzato in ambiente GNU/Linux utilizzando il linguaggio di programmazione C++.
Per la parte implementativa strettamente legata alle funzioni di cifratura è stato utilizzato il tool OpenSSL e le relative librerie di estensione per il C++.

3.2 Chiave pubblica e privata

La generazione della chiave pubblica, e della chiave privata ad essa connessa, è stata effettuata utilizzando le utility offerte da OpenSSL.
Nello specifico è stata generata una chiave RSA codificata con DES lunga 1024 bit. La chiave così generata è stata opportunamente manipolata in modo da produrre due ulteriori chiavi: chiave pubblica e chiave privata.
Le chiavi risiedono nella directory */certification* in cui in particolare troviamo:

- *pub_key.pem* - chiave pubblica del Server;
- *pri_key.pem* - chiave privata associata alla chiave pubblica del Server.

La creazione delle chiavi è modificabile a tempo di compilazione.
Anche se non è stata implementata nessuna sezione di codice in questo senso, si suppone, come già spiegato in precedenza, che la chiave pubblica sia garantita da un ente di certificazione e che la chiave privata ad essa associata sia conosciuta in maniera esclusiva dal Server.

3.3 Precisazioni introduttive

La connessione Client-Server è effettuata utilizzando le classiche funzioni per la creazione e l'utilzzo di socket. Il Server, per semplicità di gestione, non è di tipo multi-thread, ma una realizzazione in questo senso è facilmente ricavabile con piccole modifiche al codice in oggetto.
Il Server tiene memoria dei Client con cui condivide segreti ed in particolare conosce gli id dei suddetti Client e le password associate. Nel caso specifico sono state memorizzate le informazioni relative ad un solo Client, ma un'estensione di tale implementazione considerando più di un Client è facilmente realizzabile.

3.4 Connessione e scambio messaggi

Per effettuare la connessione è necessario assegnare un indirizzo IP al server e una porta associata alla connessione in apertura, e specificare la password (di almeno 8 elementi tra caratteri alfanumerici e caratteri speciali) condivisa con il Client. Il Server dopo aver settato le opportune variabili di connessione, entra in un ciclo infinito in cui risulta in attesa continua di richieste da parte del Client. Il Client per poter effettuare la connessione al Server di riferimento deve utilizzare lo stesso indirizzo IP e la stessa porta assegnate in precedenza al Server, in aggiunta alla password condivisa. Naturalmente per eseguire l'applicazione in locale è indispensabile assegnare come indirizzo IP l'indirizzo di *localhost* 127.0.0.1.
La fase immediatamente successiva a quella della connessione è quella relativa allo scambio di messaggi che viene realizzata tramite l'utilizzo di primitive *send/receive*. I primi due messaggi in chiaro sono implementati utilizzando due strutture dati opportunamente definite e allocate sia nel Client che nel Server. I restanti due messaggi, cifrati uno con la chiave pubblica e uno con la chiave di sessione, sono stringhe dinamiche create concatendando le varie componenti del messaggio discriminate da un separatore, nel caso specifico il carattere di spazio. Il separatore risulta fondamentale per poter effettuare l'analisi del messaggio da parte del ricevente dello stesso.

3.5 Controlli

Nella prima parte dell'applicazione, sia lato Server che lato Client, vengono effettuati due controlli precedenti la creazione dei socket: il primo sulla password inserita che deve essere composta da almeno 8 caratteri alfanumerici e/o caratteri speciali, il secondo sulla porta utilizzata per la linea di comunicazione che non deve appartenere al range [0, 1023] (Well Known Port) o deve essere utilizzata con molta attenzione nel caso appartenga al range [49152, 65535] (Dynamic and/or Private Port).
Lato Server, alla ricezione del messaggio M1, viene effettuato un controllo per verificare che l'identificativo del Client che ha stimolato l'inizio del protocollo sia presente tra quelli noti al Server (Client con cui il Server condivide un segreto).
Sempre lato Server, una volta ricevuto e decriptato il messaggio M3, viene effettuato un controllo per verificare che la password contenuta nel messaggio coincida con quella attesa e che il nonce contenuto nel messaggio coincida con quello atteso.
Un controllo analogo sulla password viene affettuato lato Client una volta ricevuto e decriptato il messaggio M4.

3.6 Note sul nounce

Il nonce viene creato come un intero random convertito in stringa quando necessario.

3.7 Note sulla chiave di sessione

La chiave di sessione è composta generando tre interi random che, dopo essere stati convertiti in stringhe, vengono concatenati tra loro.
La chiave viene utilizzata per effettuare cifratura a blocchi in modalità ECB. Il testo in chiaro viene gestito 64 bit per volta; ognuno dei blocchi di 64 bit viene cifrato con la stessa chiave. Per messaggi più lunghi di 64 bit, si procede suddividendo il messaggio in blocchi di 64 bit, settando a 0, se necessario, i restanti bit nell'ultimo blocco.
Il testo in chiaro, quindi, viene suddiviso in blocchi di 8 byte e viene cifrato blocco per blocco. Naturalmente anche in fase di decodifica viene effettuata un'analisi blocco per blocco sul messaggio cifrato.

3.8 Funzioni di cifratura

3.8.1 Cifratura con chiave pubblica e chiave privata

Per effettuare la cifratura dei messaggi tramite chiave pubblica e la decifratura tramite chiave privata sono state utilizzate le funzioni appartenenti alla librerie *openssl/pem.h* e *openssl/rsa.h*.

Sinossi delle funzioni utilizzate:

- RSA *PEM_read_RSAPublicKey(BIO *bp, RSA **x, pem_password_cb *cb, void *u);
- RSA *PEM_read_RSAPrivateKey(FILE *fp, RSA **x, pem_password_cb *cb, void *u);
- int RSA_public_encrypt(int flen, unsigned char *from, unsigned char *to, RSA *rsa, int padding);
- int RSA_private_decrypt(int flen, unsigned char *from, unsigned char *to, RSA *rsa, int padding);

3.8.2 Cifratura con chiave di sessione

Per effettuare la cifratura dei messaggi tramite chiave di sessione sono state utilizzate le funzioni appartenenti alla libreria *openssl/blowfish.h*.

Sinossi delle funzioni utilizzate:

- void BF_set_key(BF_KEY *key, int len, const unsigned char *data);
- void BF_ecb_encrypt(const unsigned char *in, unsigned char *out, BF_KEY *key, int enc);

4 Test

Per testare il corretto funzionamento del protocollo è stato effettuato un piccolo test consistente nell'invio, da parte del Client, di un file di testo cifrato con la chiave di sessione.
Nello specifico il Client preleva dalla directory /test un file di testo, lo memorizza in una stringa e invia la stringa al Server dopo averla opportunamente cifrata con la chiave di sessione.
Il Server riceve la stringa cifrata contenente il file di test, la decifra e stampa il contenuto della stessa su terminale.
Confrontando il contenuto del file di testo e ciò che viene stampato sul Server è facile determinare se il file è stato correttamente cifrato dal Client e decifrato dal Server.

5 Compilazione ed esecuzione

Per compilare l'applicazione posizionarsi nella directory principale del programma e digitare da shell:

- *make server* - nel caso si voglia compilare la parte relativa al server;
- *make client* - nel caso si voglia compilare la parte relativa al Client;
- *make* - nel caso si voglia compilare il codice sorgente per intero.

Per eseguire il Server posizionarsi nella directory */bin* e digitare da shell:
./server indirizzo_IP_server porta password_condivisa

Esempio:
./server 127.0.0.1 1234 ?123456789!

Per eseguire il Client posizionarsi nella directory */bin* e digitare da shell:
./client indirizzo_IP_server porta password_condivisa

Esempio:
./client 127.0.0.1 1234 ?123456789!

6 Scripting

6.1 Doppio Script

Per facilitare l'esecuzione dell'applicazione sono stati creati due script.
Sarà, quindi, sufficiente eseguire in due terminali differenti lo script relativo al Server e lo script relativo al Client.
I due script risiedono nella directory */script*.
Gli script sono stati generati in maniera tale da poter facilmente modificare a tempo di esecuzione:

- indirizzo IP del Server;
- porta associata alla connessione;
- segreto condiviso tra Client e Server.

Per eseguire il Server posizionarsi nella directory */script* e digitare da shell: *./s*

Per eseguire il Client posizionarsi nella directory */script* e digitare da shell: *./c*

6.2 Script Unico

Un'ulteriore ottimizzazione ai fini dell'esecuzione dell'applicazione è stata ottenuta generando uno script unico.
Lo script IDE (I DO EVERITHING) si occupa di mandare in esecuzione il Server, attendere un tempo pari a due secondi e quindi mandare in esecuzione il Client. I due output verranno salvati in due logfile all'interno della directory */log* con il nome rispettivamente di *log_server* e *log_client*.
In questa maniera si renderà necessario aprire una sola finestra di shell per mandare in esecuzione contemporaneamente Server e Client e sarà possibile controllare il corretto funzionamento del programma, fase di test compresa, andado a consultare il file *log_server* all'interno della directory */log*.
Al termine della sua esecuzione lo script chiuderà automaticamente i processi Server e Client.
Anche questo script è stato creato in maniera tale da poter facilmente modificare a tempo di esecuzione:

- indirizzo IP del Server;
- porta associata alla connessione;

- segreto condiviso tra Client e Server.

Per eseguire lo script posizionarsi nella directory */script* e digitare da shell: *./ide*

Per visualizzare i file di log posizionarsi della directory */log* e digitare da shell:
cat log_server - per visualizzare il log del Server
cat log_client - per visualizzare il log del Client