Azzolini Riccardo 2021-04-29

Gestione delle chiavi simmetriche

1 Gestione delle chiavi

Quando si usa la crittografia, sia simmetrica che asimmetrica, si ha il problema di come gestire, e in particolare distribuire, le chiavi:

- le chiavi simmetriche devono essere condivise in modo segreto solo da chi deve usarle;
- per le chiavi pubbliche non serve la segretezza, ma è invece importante dare una prova dell'associazione tra una chiave e l'identità dell'utente a cui essa appartiene.

Dati i diversi requisiti della distribuzione delle chiavi simmetriche e di quelle pubbliche, le soluzioni nei due casi applicate sono diverse. Adesso verranno presentate le principali soluzioni, a partire da quelle per le chiavi simmetriche.

2 Gestione delle chiavi simmetriche

Tipicamente, una chiave simmetrica è associata a un canale (o a una sessione) di comunicazione tra due utenti/entità, è condivisa da solo due utenti che la usano per comunicare in modo sicuro tra di loro. Allora, per gestire tutte le possibili comunicazioni sicure tra n utenti servono almeno

$$\binom{n}{2} = \frac{n!}{2!(n-2)!} = \frac{n(n-1)}{2}$$

chiavi simmetriche (una per ogni coppia non ordinata di utenti[1]), che è un numero molto grande (quadratico nel numero n di utenti).

La più semplice soluzione per la gestione delle chiavi sarebbe affidare a un amministratore il compito di distribuire a ogni utente, ovvero configurare manualmente su ogni macchina, tutte le chiavi necessarie, ma ciò è assolutamente non scalabile:

- se una rete ha un numero significativo di utenti/nodi le chiavi da distribuire sono troppe, soprattutto considerando che non basta distribuirle una volta, ma bisogna periodicamente sostituirle per evitare che ci sia tempo di ottenerle mediante attacchi a forza bruta mentre sono ancora in uso;

[1] Si considerano le coppie non ordinate di utenti perché una singola chiave simmetrica è sufficiente a consentire la comunicazione tra due utenti in entrambe le direzioni.

- a ogni cambiamento della topologia della rete, ovvero a ogni inserimento o rimozione di un $(n+1)$-esimo utente nel gruppo, è necessario distribuire o ritirare una chiave a ciascuno degli n altri utenti.

Servono perciò dei protocolli *sicuri* per automatizzare la generazione e lo scambio di chiavi simmetriche.

I protocolli per la gestione delle chiavi simmetriche possono essere classificati in due famiglie, in base al servizio che offrono:

- **key transport**: la chiave simmetrica viene generata dalle due parti comunicanti, oppure da un'autorità esterna (**KDC**, **Key Distribution Center**), poi si utilizza un protocollo per trasportare tale chiave alle parti interessate in modo sicuro, impedendo che altri possano leggerla o modificarla / sostituirla;
- **key agreement**: la chiave viene generata tramite un protocollo che coinvolge entrambe le parti, le quali si scambiano delle informazioni pubbliche, non riservate, ed eseguono delle computazioni locali indipendenti tramite le quali arrivano a ottenere lo stesso valore della chiave (si "mettono d'accordo" sulla chiave, da cui il termine "agreement"), senza mai trasportare la chiave stessa.

Nel seguito verranno presentati:

- il protocollo di key transport *Needham-Schroeder*, nelle sue versioni a chiave pubblica e a chiave segreta;
- il protocollo di key agreement *Diffie-Hellman*.

3 Protocollo semplice di key transport

Prima di introdurre il protocollo Needham-Schroeder è utile iniziare da un protocollo di key transport semplice ma non del tutto sicuro, basato sulla cifratura asimmetrica. Se due utenti A e B vogliono condividere una chiave segreta, tale protocollo prevede che la chiave sia generata da uno dei due utenti, ad esempio B, e condivisa con l'altro utente A cifrandola con la chiave pubblica di A. Più nel dettaglio, il protocollo funziona in questo modo:

1. A genera una coppia di chiave pubblica KP_A e chiave privata KR_A, e invia la chiave pubblica a B insieme a un suo identificatore[2] ID_A.
2. B genera una chiave di sessione (simmetrica) SK_{AB} e la invia ad A insieme al suo identificatore ID_B, cifrando il tutto con la chiave pubblica KP_A che ha appena ricevuto.

[2] Non è specificato come debbano essere fatti gli identificatori degli utenti; ad esempio, essi potrebbero essere indirizzi IP, o altro.

3. A usa la sua chiave privata KR_A per decifrare il messaggio ricevuto da B, ottenendo così la chiave di sessione SK_{AB}, che successivamente usa per comunicare in modo sicuro con B.

I passi del protocollo possono essere descritti in forma più schematica scrivendo

(1) $A \to B$: ID_A, KP_A
(2) $B \to A$: $\{SK_{AB}, ID_B\}_{KP_A}$
(3) $A \to B$: $\{\text{dati}\}_{SK_{AB}}$

dove la notazione $\{M\}_K$ è un altro modo di indicare la cifratura del messaggio M con la chiave K, che può forse risultare maggiormente leggibile rispetto a $E_K(M)$ quando M è complesso, composto dalla concatenazione di diversi valori. Si noti che i valori concatenati vengono cifrati *tutti insieme*, non ciascuno separatamente: ciò è di fondamentale importanza, in quanto impedisce a un attaccante di sostituire solo alcune parti del messaggio (ad esempio con parti di altri messaggi intercettati in precedenza) senza bisogno di decifrarlo e ricifrarlo.

3.1 Vulnerabilità

Il protocollo così definito *non è sicuro* perché è vulnerabile a un attacco **man-in-the-middle**, un tipo di attacco nel quale un attaccante E si mette "in mezzo" tra A e B e prende il controllo della comunicazione, riuscendo a leggere e modificare i messaggi senza che A e B se ne accorgano.

Quando l'utente A invia il primo messaggio del protocollo,

(1) $A \to B$: ID_A, KP_A

l'attaccante E lo intercetta, genera una sua coppia di chiavi KP_E e KR_E, e invia il messaggio a B sostituendo la chiave pubblica di A con la propria:

(1) $A \to B$: ID_A, KP_E

B non ha modo di accorgersi che la chiave ricevuta non è l'originale di A, quindi usa normalmente tale chiave per cifrare il messaggio contenente la chiave di sessione:

(2) $B \to A$: $\{SK_{AB}, ID_B\}_{KP_E}$

E intercetta anche questo messaggio, che (grazie alla sostituzione della chiave effettuata prima) può decifrare con la propria chiave privata KR_E per ottenere la chiave segreta SK_{AB}. Poi, E usa la chiave pubblica KP_A, che conosce dal messaggio (1), per ricifrare il messaggio SK_{AB}, ID_B decifrato, ottenendo $\{SK_{AB}, ID_B\}_{KP_A}$, che invia ad A:

(2) $B \to A$: $\{SK_{AB}, ID_B\}_{KP_A}$

Così, A riceve esattamente il messaggio che si aspetta, lo decifra con la propria chiave privata KR_A, e inizia a comunicare usando SK_{AB}, senza accorgersi di nulla,

(3) $A \to B$: $\{\text{dati}\}_{SK_{AB}}$

ma E è in possesso della chiave segreta, quindi può decifrare i messaggi scambiati con SK_{AB} per leggerli, eventualmente modificarli e ricifrarli, ecc.

L'attacco è possibile perché non si ha la prova che la chiave pubblica contenuta nel messaggio (1) sia effettivamente quella dell'utente A. Come già detto, questo sarà proprio il problema da risolvere nell'ambito della gestione delle chiavi pubbliche.

4 Needham-Schroeder con chiave pubblica

Il protocollo **Needham-Schroeder con chiave pubblica** è un protocollo di key transport che usa le chiavi pubbliche di due utenti A e B per scambiare tra loro una chiave simmetrica generata da uno dei due. Esso introduce un server S, chiamato **autorità di distribuzione**, che si occupa di distribuire le chiavi pubbliche. Per l'esecuzione del protocollo è necessario che S conosca le chiavi pubbliche KP_A e KP_B e che A e B si fidino di S.

4.1 Versione non sicura

Una prima versione del protocollo, non sicura ma utile a scopo introduttivo, è la seguente.

Per iniziare, A manda a S un messaggio contenente il suo identificatore ID_A e l'identificatore ID_B dell'utente con cui vuole comunicare:

(1) $A \to S$: ID_A, ID_B

S risponde inviando ad A la chiave pubblica dell'utente B, insieme all'identificatore ID_B per confermare che la chiave appartiene a B, il tutto cifrato con la chiave privata KR_S del server (e di conseguenza decifrabile con la chiave pubblica KP_S, che si suppone che A conosca).

(2) $S \to A$: $\{KP_B, ID_B\}_{KR_S}$

Tale cifratura realizza l'autenticazione, cioè garantisce che questo messaggio provenga da S, quindi se A si fida di S ha la prova che KP_B sia effettivamente la chiave pubblica di B.

Se il messaggio (2) contenesse solo KP_B, e non anche ID_B, sarebbe possibile un **attacco a replay**: tale messaggio potrebbe essere sostituito con un altro messaggio precedentemente restituito da S, contenente la chiave di un diverso utente, senza che A se ne accorga. Ad esempio, un attaccante E e un utente F che collude, è d'accordo con E potrebbero realizzare il seguente attacco:

1. F dice di voler comunicare con E, ottenendo così dal server il messaggio $\{KP_E\}_{KR_S}$, che poi passa a E.
2. Successivamente, E intercetta il messaggio $\{KP_B\}_{KR_S}$ in un'altra esecuzione del protocollo e lo sostituisce con $\{KP_E\}_{KR_S}$.
3. Nei passi successivi del protocollo A comunicherà con B usando KP_E invece di KP_B, perciò E potrà decifrare i messaggi.

Invece, siccome i messaggi di risposta di S contengono anche gli identificatori, se si provasse un attacco del genere A troverebbe ID_E al posto di ID_B, dunque si accorgerebbe di aver ricevuto una chiave pubblica associata a un utente diverso da quello con cui vuole comunicare.

Tornando all'esecuzione del protocollo, il messaggio successivo è

(3) $A \to B$: $\{N_a, ID_A\}_{KP_B}$

con il quale A comunica a B il proprio identificatore (in modo che B sappia con chi sta comunicando) e invia un **nonce** N_a, un valore casuale usato una volta sola (se ne rigenera uno diverso ogni volta). Il tutto è cifrato con la chiave pubblica di B, in modo che solo B possa leggerlo. Il nonce N_a servirà più avanti nel protocollo per dare ad A una prova dell'identità di B. Infatti, il protocollo Needham-Schroeder non esegue solo il semplice scambio di una chiave, ma anche l'**autenticazione del peer**, cioè dell'utente con cui si sta comunicando.

Successivamente, B interagisce con S per ottenere la chiave pubblica di A, nello stesso modo in cui A ha ottenuto quella di B:

(4) $B \to S$: ID_B, ID_A
(5) $S \to B$: $\{KP_A, ID_A\}_{KR_S}$

Poi, B prende il nonce N_a che aveva ottenuto decifrando il messaggio (3), genera un altro nonce N_b, e invia entrambi ad A in un messaggio cifrato con la chiave KP_A appena ottenuta:

(6) $B \to A$: $\{N_a, N_b\}_{KP_A}$

Quando A riceve questo messaggio, lo decifra con KR_A e verifica che N_a sia effettivamente uguale al nonce che aveva inviato nel messaggio (3). Se ciò è vero, A ha la prova che l'utente con cui sta attualmente comunicando sia riuscito a decifrare il messaggio (3), e tale decifratura richiede la chiave privata KR_B di cui solo B è in possesso, quindi A ha la prova di stare comunicando con B *in questa sessione*, in questo momento. Infatti, essendo ogni volta diverso, N_a fornisce una garanzia di "freshness" dei messaggi successivi, conferma che essi sono stati generati in risposta al (3), escludendo ad esempio la possibilità di un attacco a replay in cui un attaccante si finge B riproducendo i messaggi di una vera esecuzione del protocollo avvenuta in precedenza tra A e B.

Invece, il nonce N_b presente nel messaggio (6) viene usato in modo analogo per realizzare l'autenticazione nell'altra direzione: siccome solo A ha la chiave privata KR_A necessaria

per decifrare il (6) e ottenere N_b, rimandando il valore di N_b a B (cifrato con KP_B in modo che non possa essere letto o modificato da altri)

(7) $A \to B$: $\{N_b\}_{KP_B}$

dà a B la prova di stare attualmente comunicando con il vero utente A. Complessivamente si ha dunque una **doppia autenticazione**, un'autenticazione in due direzioni tra A e B.

Dopo il passo (7), B può generare una chiave simmetrica e spedirla ad A cifrandola con KP_A.

4.2 Vulnerabilità

Il protocollo così definito è vulnerabile ad attacchi di tipo man-in-the-middle. In particolare, se un attaccante E convince in qualche A a iniziare con lui una sessione (un'esecuzione corretta del protocollo, nella quale A sa che sta comunicando con E), allora può sfruttare i messaggi ricevuti da A per comunicare con B fingendo di essere A (qui E non intercetta una comunicazione tra A e B, ma può ad esempio interagire con B in modi in cui A è autorizzato a farlo mentre E non lo è). Sia nella sessione tra A ed E che in quella tra E e B le parti di comunicazione con il server S rimangono invariate, identiche a quelle di una normale sessione, dunque nel seguito verranno omesse.

L'attacco inizia con i primi passi di una normale sessione tra A ed E, tra cui in particolare:

(1) $A \to E$: $\{N_a, ID_A\}_{KP_E}$

Poi, per iniziare una sessione con B fingendo di essere A, E decifra questo messaggio, lo ricifra con KP_B (che ha ottenuto normalmente da S) e lo invia a B:

(2) $E \to B$: $\{N_a, ID_A\}_{KP_B}$

B decifra questo messaggio con la propria chiave privata e, dopo aver ottenuto la chiave KP_A da S, invia a E (che pensa sia A) il messaggio con il nonce N_a estratto dal (2) e il nuovo nonce N_b, il tutto cifrato con KP_A:

(3) $B \to E$: $\{N_a, N_b\}_{KP_A}$

A questo punto, per convincere B di essere A, E deve rimandare a B il valore N_b decifrato con KR_A e ricifrato con KP_B, ma non ha la chiave privata KR_A per decifrare. Intanto, però, nella sessione tra A ed E l'utente A sta aspettando proprio di ricevere un messaggio $\{N_a, N_e\}_{KP_A}$. E se ne approfitta mandando invece il messaggio $\{N_a, N_b\}_{KP_A}$:

(4) $E \to A$: $\{N_a, N_b\}_{KP_A}$

A non ha modo di accorgersi che il messaggio proviene da B invece che da E, perché:

- il valore del nonce N_a è effettivamente quello inviato nel messaggio (1), grazie al fatto che E ha inoltrato a B tale valore nel messaggio (2);
- siccome un nonce è un semplice numero casuale, A non può sapere che N_b è stato generato da B e non da E.

Di conseguenza, A procede normalmente, decifrando il messaggio con la propria chiave privata e rimandando ad E il valore del nonce N_b, questa volta cifrato con KP_E:

(5) $A \to E$: $\{N_b\}_{KP_E}$

Adesso E può decifrare il (5) con KR_E per ottenere finalmente il valore del nonce N_b che non era riuscito a decifrare prima. Infine, E ricifra N_b con KP_B e lo manda a B per completare l'esecuzione del protocollo con B:

(6) $E \to B$: $\{N_b\}_{KP_B}$

Da questo momento in poi, E può comunicare con B mentre B pensa di essere certo di comunicare con A, avendo ricevuto come "prova" il valore N_b decifrato con KR_A e ricifrato.

4.3 Versione corretta

L'attacco appena presentato è possibile perché l'attaccante E riesce a inoltrare il messaggio $\{N_a, N_b\}_{KP_A}$ da B ad A senza che A si accorga che tale messaggio proviene da un utente diverso dall'E con cui sta comunicando. Per prevenire l'attacco è allora sufficiente inserire in tale messaggio l'identità dell'utente B che l'ha generato: $\{N_a, N_b, ID_B\}_{KP_A}$. Così, nell'attacco, al passo (4) A si aspetta di ricevere $\{N_a, N_e, ID_E\}_{KP_A}$, ma invece riceve $\{N_a, N_b, ID_B\}_{KP_A}$, nel quale l'identificatore dell'utente non coincide con quello previsto, dunque si accorge dell'attacco (e può interrompere la comunicazione con E per impedire il completamento dell'attacco).

In generale, aggiungere gli identificatori degli utenti ai messaggi è importante per impedire l'inoltro / il riutilizzo di tali messaggi in sessioni con altri utenti.

Riassumento, una normale esecuzione della versione corretta del protocollo consiste nei seguenti passi (seguiti dallo scambio di una chiave di sessione simmetrica e dalle successive comunicazioni tra A e B cifrate con tale chiave):

(1) $A \to S$: ID_A, ID_B
(2) $S \to A$: $\{KP_B, ID_B\}_{KR_S}$
(3) $A \to B$: $\{N_a, ID_A\}_{KP_B}$
(4) $B \to S$: ID_B, ID_A
(5) $S \to B$: $\{KP_A, ID_A\}_{KR_S}$
(6) $B \to A$: $\{N_a, N_b, ID_B\}_{KP_A}$
(7) $A \to B$: $\{N_b\}_{KP_B}$

(qui l'ID aggiunto per prevenire l'attacco man-in-the-middle è evidenziato in rosso).