Comunicazione sicura: i protocolli crittografici

Definizione del contesto

Comunicazione sicura lungo un canale insicuro

Indichiamo con la parola **canale** il mezzo tramite il quale due entita' possono comunicare. Quando il canale e' accessibile anche ad altre entita', oltre a quelle che vogliono comunicare tra di loro, allora il canale si dice **insicuro**. Poiche' internet e' un canale diviso tra moltissimi utenti, e' definitivamente un canale insicuro.

Nella comunicazione tra persone, browser/server, online banking client/server, router, e' necessario che le informazioni scambiate tra le due entita' siano **protette** e che queste identita' siano **autenticate** per verificarne l'identita'.

Comunicazione sicura

Per fare in modo che una comunicazione sia sicura, attraverso un canale insicuro come Internet dobbiamo stabilire delle proprieta' di sicurezza che devono essere garantite durante la comunicazione:

- **Confidenzialita'**: solo Alice e Bob devono poter "capire" il contenuto dei messaggi;
- **Integrita'**: i messaggi non devono venire modificati;
- **Autenticazione**: Alice e Bob devono essere sicuri che il loro interlocutore sia effettivamente chi dice di essere (e che loro si aspettano);
- **Non-repudiation**: Alice non puo' negare di aver spedito un messaggio a Bob.

Autenticazione nella rete

Quando si effettuano comunicazioni nella rete e' necessario specificare cosa si intenda autenticare.

Entity Authentication

Garantisce l'identita' dell'utente che in tempo reale sta comunicando all'interno di una specifica sessione di protocollo.

Message Authentication

Garantisce che il messaggio ricevuto proviene dalla sorgente indicata, senza dare alcuna indicazione di quando il messaggio sia stato effettivamente creato.

Tipologia di attaccante

L'attaccante (Trudy) e' un **host malevolo** connesso ad Internet che puo' voler modificare il normale comportamento dei protocolli per scopi illeciti.

Protocolli TCP/IP

Alice e Bob sono due entita' che utilizzano Internet come canale di comunicazione, quindi usano i protocolli dello stack TCP/IP.

n	Internet	ISO/OSI 7 layer model	n
		Application	7
5	Application	Presentation	6
		Session	5
4	TCP	Transport	4

n	Internet	ISO/OSI 7 layer model	n
3	IP	Network	3
2	Network Interface	Data Link>	2
1	Hardware	Physical	1

> I protocolli di comunicazione definiscono il formato, l'ordine dei messaggi spediti e ricevuti tra le entita' della rete, e le azioni intraprese dopo la trasmissione/ricezione

Cosa puo' fare un attaccante

In genere un attaccante puo' essere definito **attivo**/**passivo** a seconda di cosa puo' fare lungo un canale di comunicazione. Rimanendo in ascolto lungo il canale puo' intercettare il messaggio (minacciando la confidenzialita') per poi:

- Leggere solamente il messaggio (passivo);
- Effettuare azioni sul messaggio (attivo):
  - Cancellare il messaggio senza inoltrarlo al destinatario;
  - Modificare i messaggi in transito (minaccia all'integrita');
  - Puo' spedire messaggi arbitrari (minaccia all'autenticazione);
  - Puo' riutilizzare vecchi messaggi salvati, tramite un **replay attack** (minaccia all'autenticazione).

Contromisure

Per impedire a Trudy di modificare il funzionamento del protocollo, e' necessario crittografare il messaggio (tramite delle **primitive crittografiche**) per continuare a garantire la confidenzialita' e fare in modo che solo Alice e Bob lo possano decifrare.

La segretezza puo' pero' non essere sufficiente per garantire anche l'autenticazione e l'integrita'.

Per fare in modo che un protocollo sia del tutto sicuro, e dunque vengano garantite tutte le proprieta' di sicurezza, sara' necessario usare una combinazione di primitive crittografiche e costruire un **protocollo di sicurezza**.

Introduzione alla crittografia

Storia della crittografia

Proprieta' degli algoritmi di cifratura

Alice vuole mandare un messaggio m a Bob, mentre Trudy e' l'attaccante passivo (puo' leggere il messaggio M):

- Alice da in input all'**algoritmo di codifica** la chiave di codifica insieme al messaggio in chiaro, ottenendo il messaggio cifrato;
- Alice spedisce il messaggio cifrato a Bob attraverso il canale;
- Bob applica l'**algoritmo di decodifica** con la chiave di decodifica insieme messaggio cifrato, ottenendo il messaggio in chiaro.

Principio di Kerchoff

La proprieta' di un buon algoritmo di cifratura/decifratura e' sintetizzata dal **principio di Kerchoff**:

> La sicurezza e' data dalla robustezza dell'algoritmo e dalla segretezza della chiave, non della segretezza dell'algoritmo.

Per questo, nel caso precedente, Trudy puo' avere accesso al canale, dunque ai messaggi, ed essere a conoscenza dell'algoritmo di cifratura, in quanto la sicurezza e' garantita dal possesso delle chiavi di cifratura/decifratura.

In particolare valgono le seguenti affermazioni:

- **E**: funzione codifica;

- **D**: funzione decodifica;
- **m**: messaggio;
- **x**: messaggio cifrato;
- **K**: chiave per la cifratura;
- **K'**: chiave per la decifratura.

> $D_{K'}(E_K(m)) == m$

> $E_K(m)$ e' semplice da calcolare, dati m e K

> $D_{K'}(x)$ e' semplice da calcolare, dati x e K', ma per Trudy deve essere molto difficile da calcolare (in quanto non possiede K')

> Se $x == E_K(m)$, e' difficile trovare m senza K

Lunghezza delle chiavi

Per quanto possa essere valido un algoritmo crittografico, esso puo' essere forzato provando a decifrare il messaggio con tutte le chiavi possibili tramite un attacco bruteforce.

Per questo motivo, il numero delle chiavi possibili deve essere grande a sufficienza per rendere computazionalmente impossibile questo attacco. Se scelgo una chiave di *n* bit, lo spazio delle chiavi sara' 2^n.

Gestione di messaggi molto lunghi

Block ciphers

Algoritmo che suffivide il messaggio in blocchi di lunghezza predefinita e li cifra/decifra uno alla volta con la stessa chiave (aggiungendo dei bit finali all'ultimo blocco). Tipicamente un blocco e' di 64 bit.

Stream ciphers

Algoritmo che cifra il messaggio 1 bit o 1 byte alla volta usando una sequenza denominata **keystream** generata randomicamente a partire dalla chiave o dal messaggio.

Crittografia a chiave simmetrica e asimmetrica

Crittografia simmetrica

Se si utilizza la stessa chiave per la codifica e la decodifica allora si tratta di crittografia a chiave simmetrica o a chiave segreta: infatti la segretezza e l'integrita' sono garantite proprio da quest'ultima.

Se Trudy venisse a conoscenza della chiave segreta, potrebbe:

- Produrre nuovi messaggi a nome di Alice o di Bob;
- Leggere tutti i messaggi trasmessi;

Le implementazioni di algoritmi per la crittografia simmetrica sono in genere molto veloci (esistono **implementazioni hardware**) e usano chiavi relativamente corte (64-256 bit), inoltre possono essere utilizzati per la generazione di numeri casuali.

Gli svantaggi sono legati alla gestione e la distribuzione delle chiavi, in quanto:

- La chiave deve essere segreta, e rimanere tale;
- E' necessario che le parti che devono comunicare condividano a priori la chiave segreta da utilizzare per la comunicazione;
- Per evitare gli attacchi bruteforce e' utile cambiare spesso la chiave riproducendo pero' il problema della distribuzione.

La crittografia simmetrica viene utilizzata in diversi ambiti e con diversi obiettivi, in particolare per garantire confidenzialita' e integrita' con il problema di dover **scambiare prima la chiave lungo un canale sicuro**. Viene usata anche per la memorizzazione sicura su media insicuri (filesystem criptati) e per l'autenticazione con protocollo challenge-response.

Gestione delle chiavi

Nel caso in cui ci siano *n* utenti che vogliono parlare tra di loro in modo sicuro,ciascuna coppia di utenti ha bisogno di una chiave il che porta ad un numero polinomiale (n(n-1)/2) di chiavi che si devono scambiare in modo sicuro mediante:

- Scambio fisico;
- Canale sicuro. Poiche' non e' certo che tutti gli *n* utenti parlino tra di loro, l'algoritmo e' inefficiente.

KDC - Key Distribution Center

Il Key Distribution Center e' da considerarsi la *n+1*esima **parte fidata** in un gruppo di *n* utenti, e si occupa di **generare le chiavi** per ogni coppia che le richiede (questo presuppone che gli utenti debbano condividere a priori una chiave con il KDC, per un totale di *n* chiavi totali iniziali). Con la tecnica del KDC si ha un algoritmo piu' efficiente in quanto il numero di chiavi da scambiare inizialmente non e' polinomiale al numero di utenti totali.

I problemi sono:

- Il KDC deve essere fidato;
- Il KDC diventa un collo di bottiglia e unico punto di fallimento;
- Distribuire inizialmente le chiavi tra gli *n* utenti e il KDC.

Se l'utente U_1 vuole parlare con l'utente U_2:

- U_1 -> KDC: "Collegami a U_2";
- KDC genera la nuova chiave $K_{1\text{-}2}$;
- KDC -> U_1: $E_{K_1}(K_{1\text{-}2})$;
- KDC -> U_2: $E_{K_2}(K_{1\text{-}2}$, "U_1 vuole parlare con te").

Algoritmi a chiave simmetrica famosi

- DES - Data Encryption Standard;
  - Triple DES (3DES);
- DES-X;
- AES - Advanced Encryption Standard;
- IDEA - International Data Encryption Algorithm;
- RC2;

Crittografia asimmetrica

La crittografia asimmetrica (detta anche crittografia a chiave pubblica) utilizza invece due chiavi, una per la codifica, l'altra perl a decodifica.

Ogni utente ha una coppia di chiavi, dunque non e' necessario lo scambio di chiavi iniziale:

- Chiave pubblica: informazione da diffondere;
- Chiave segreta: segreto da custorire.

Tramite la crittografia asimmetrica:

- La segretezza del messaggio e' garantita;
- Sono necessarie meno chiavi, ma e' importante garantire l'autenticita' delle chiavi pubbliche;
- E' possibile garantire integrita' e autenticazione procedendo al contrario (crittando con la propria privata), ma in questo modo la segretezza non e' piu' garantita in quanto chiunque possieda la chiave pubblica puo' leggere.

Tendenzialmente la crittografia asimmetrica, rispetto a quella simmetrica, e' piu' lenta e usa chiavi piu' lunghe (1024-2048 bit), qunidi inefficienti per messaggi lunghi.

> La best practice prevede l'utilizzo di chiavi asimmetriche per lo scambio di chiavi simmetriche.

Algoritmi a chiave asimmetrica famosi

- RSA
- Crittosistema di Rabin
- ElGamal

Funzioni hash

La funzione hash e' una funzione computazionalmente efficiente che mappa stringhe binarie di lunghezza arbitraria in stringhe di lunghezza costante dette **hash value** (o **message digest**, **checksum**, **one-way function**):

> h: $\{0,1\}^*$ -> $\{0,1\}^n$

Per definizione deve avere le seguenti proprieta':

- **Computazionalmente efficiente**, dunque semplice da calcolare:
  - Dato *x*, e' facile da calcolare *h(x)*;
- In grado di effettuare **compressione**:
  - *h* mappa input di lunghezza arbitraria ad output *h(x)* di lunghezza *n* definita;
- **Collision resistant**:
  - Difficile trovare *m* e *m'* tali che *h(m)=h(m')*;
- **Irreversibile** (one-way):
  - **Preimage resistance**: dato *y* e' difficile trovare *x*, tali che *y=h(x)*;
  - **Second preimage resistance**: dato *m* e' difficile trovare *m'*, tali che *h(m)=h(m')*;
- **Effetto valanga**:
  - Una piccola modifica al valore di *m* deve alterare tutto *h(m)*.

Paradosso del compleanno

La probabilita' che in un gruppo di *n* persone ne esistano almeno 2 che sono ante nello stesso giorno aumenta velocemente con *n*, piu' di quanto potrebbe dire l'intuito.

Lo stesso paradosso si puo' applicare alla probabilita' di collisione nelle funzioni di hash, che dipende dalla lunghezza *n* del valore di hash:

- Sia *y* un hash lungo *n* bit, il numero di tentativi per trovare un *x* tale che *h(x)=y* e' di 2^{n-1};
- Dato un insieme di hash value lunghi *n* bit, un insieme di almeno $2^{n/2}$ input ha un'alta probabilita' di contenere una coppia che causi collisione.

Utilizzi

Password hashing

Non si memorizzano le password in chiaro, ma l'hash delle password (preferibilmente concatenate con il salt, come accade nei sistemi Unix). Sono comunque possibili attacchi offline nel caso si abbia a disposizione il database.

Message digesting

Le funzioni hash vengono usate anche per accorciare messaggi lunghi, in particolare nei contesti della firma digitale con chiave asimmetrica e della sincronizzazione dei file via rete (rsync).

Nell'ambito dei protocolli crittografici risulta particolarmente utile in quanto ha un effetto collaterale positivo: unisce e fonde insieme le parti del messaggio che viene dato in input alla funzione di hash.

Garantire integrita' dei dati

Si suppone si voglia proteggere un dato *x*, si calcola il suo hash *h(x)* e si archivia in un luogo protetto. In seguito, per vedere se il dato *x* e' stato modificato, si calcola nuovamente il suo valore di hash e lo si confronta con il valore precedentemente calcolato.

Questo metodo e' utilizzato per verificare l'integrita' dei file negli HIDS (Host-based Intrusion Detection Systems) e per verificare l'integrita' dei file scaricati dalla rete.

L'unico problema di questo approccio e' che devo fare affidamento ad un sistema di archiviazione, e nel caso in cui questo abbia una falla e l'attaccante possa modificare il valore archiviato, allora l'attaccante e' in grado di inserire *h(x')* (modificato), al posto di *h(x)*.

MAC - Message Authentication Code

Si tratta di funzioni hash con chiave, che oltre a prendere in input una stringa di lunghezza arbitraria, prende anche una **chiave segreta condivisa**:

> h_k: $\{0,1\}^*$ -> $\{0,1\}^n$

In questo caso la funzione deve soddisfare una proprieta' addizionale detta **computation resistance**:

> Per ogni possibile chiave *k* non nota all'avversario e un insieme di valori di stringhe di input, con il rispettivo valore di hash *(x_i, $h_k(x_i)$)*, deve essere computazionalmente impossibile calcolare l'hash di input diversi $h_k(x)$ per ogni nuovo input *x*.

La computation resistance garantisce integrita' e autenticazione (**data origin**) del messaggio.

Il MAC non garantisce segretezza (in quanto non viene spedito anche il messaggio in chiaro, di modo che il destinatario ricalcoli il MAC) o non-repudiation (anche se Bob sapesse che solo Alice potrebbe aver calcolato il MAC del messaggio ricevuto, Bob non e' in grado di dimostrare che puo' non essere stato lui a calcolare l'hash con chiave). Inoltre non puo' essere verificato da una terza parte (solo chi possiede la chiave segreta condivisa).

Utilizzi

Anche il MAC serve a verificare l'integrita' dei dati:

- Il mittente spedisce il messaggio *m* e il MAC $h_k(m)$;
- Il destinatario riceve entrambe le parti, e a partire da *m* ricalcola il MAC $h_k(m)$;
- Se il messaggio calcolato corrisponde a quello ricevuto, il messaggio non e' stato modificato.

Nel filesystem, per garantire l'integrita' dei file, oltre al file *file.pdf*, si archivia anche $h_{pwd\_utente}(file.pdf)$. In questo modo, l'utente che ha generato il file, e' l'unico in grado di modificarlo perche' e' l'unico a conoscere la chiave che aggiorna l'hash memorizzato.

Funzioni hash famose

- MD4: a 128 bit, e' diventata obsoleta in quanto nel 1996 e' stata trovata una collisione;
- MD5: a 128 bit, e' molto diffusa anche per i protocolli di rete, ma esistono algoritmi efficienti in grado di generare stringhe che collidono;
- SHA: famiglia di 5 diverse funzioni (Secure Hash) la cui piu' diffusa e' la SHA-1 a 160 bit;
- RIPEMD: alternativa europea a MD4 e MD5.

Firma digitale

La firma digitale e' l'analogo della firma su carta, ma nel cyberspazio.

Caratteristiche

- Deve essere facilmente prodotta dal legittimo proprietario;
- Nessun utente deve poter riprodurre la firma di altri;
- Chiunque deve poter facilmente verificare una firma;
- La firma non deve essere utilizzabile su un altro documento.

Proprieta' di sicurezza da garantire

Le proprieta' di sicurezza che si vogliono garantire sono:

- Integrita': il messaggio, una volta firmato, non deve poter essere alterato;
- Autenticazione: il mittente ha sottoscritto il contenuto del messaggio;
- Non-ripudio: il firmatario non puo' rinnegare la paternita' del messaggio;

La funzione hash con chiave (MAC) garantisce sia integrita' che autenticazione del messaggio, ma non e' sufficiente in quanto non garantisce il non ripudio: Bob puo' creare un MAC(m) esattamente come Alice, quindi, anche se Bob sa che Alice e' l'autrice del messaggio, non puo' mostrarlo come prova a nessuno.

Solo chi possiede la chiave segreta condivisa puo' verificare la firma, non chiunque.

Creazione della firma

La firma digitale si ottiene basandosi sulla crittografia asimmetrica:

- La firma si ottiene **cifrando con la chiave privata**;
- La verifica della firma si ottiene **decifrando con la chiave pubblica**.

Quindi:

- Chiunque puo' verificare un messaggio firmato da Alice (se ne possiede la chiave pubblica);
- Solo Alice puo' spedire messaggi firmati;
- Non ripudio: Bob non puo' creare la firma di Alice perche' non conosce la sua chiave privata.

Le primitive crittografiche a chiave asimmetrica sono particolarmente inefficienti, per questo, invece di creare la firma sul messaggio in chiaro, viene fatta sul digest:

- Calcolo del message digest del messaggio;
- Cifratura del digest con chiave privata (si ottiene la firma digitale del messaggio);
- Invio della firma digitale e del messaggio in chiaro.

Verifica della firma

Il destinatario che riceve un messaggio che continene una firma:

- Separa il messaggio in chiaro e la firma;
- Decodifica la firma con la chiave pubblica del mittente ed ottiene il digest del messaggio ricevuto;
- Calcola il digest sul messaggio in chiaro;
- Confronta il digest appena calcolato col digest ricevuto:
  - Se coincidono allora la firma e' valida ed il messaggio e' integro;
  - Altrimenti il testo e' stato alterato, dunque si rifiuta il messaggio.

Resistenza alla contraffazione

> Sia $Firma(K^{-1}, m)$ la funzione che firma il messaggio m e $Verifica(K, x, m)$ la funzione di verifica (con $x=Firma(K^{-1}, m)$) allora:
>
> - **Non si riesce a calcolare** $Firma(K^{-1}, m)$ da m e K;
> - **Resiste all'attacco di forza bruta**: data K, non si riesce a produrre $Firma(K^{-1}, m)$ per nessun m.

Problema dell'identita'

Basandosi sulla crittografia asimmetrica, rimane il problema dell'associazione identita'-chiave pubblica: l'utilizzo della firma digitale garantisce che un messaggio e' stato firmato con una determinata chiave privata: pero' **non dice nulla sull'autore della firma** se non si e' a conoscenza in modo certo della chiave pubblica della persona.

E' necessario un modo sicuro di scambiarsi le chiavi pubbliche, garantendo (non tanto la confidenzialita') l'autenticazione. Puo' essere fatto di persona o tramite una terza parte fidata come una **Certification Authority**.

Hashing vs MAC vs Firme Digitali

Hashing (*digest* privato)

- Produce il *digest* di un messaggio;
- Per garantire integrita' deve venire memorizzato separatamente dal messaggio.

MAC (*digest* cifrato)

- Il *digest* viene protetto da una chiave condivisa e una segreta;
- Puo' venire trasmesso lungo un canale pubblico.

Firma digitale (non-repudiation)

- Il *digest* viene cifrato da una chiave privata: integrita', non ripudio e data origin authentication, non confidenzialita';
- Non ci sono dati segreti condivisi con chi verifica e chi firma.

Certificato digitale

Il certificato digitale e' l'analogo della carta d'identita' (viene emesso da un'entita' riconosciuta, associa l'identita' di una persona al suo aspetto fisico, ecc.) e risolve alcuni problemi rimasti aperti con l'impiego della crittografia asimmetrica:

- Chi garantisce che la chiave pubblica di Bob sia stata rilasciata proprio a Bob?
- Nel caso di comunicazione con un utente che non si conosce, come faccio a sapere chi possiede la chiave privata corrispondente a quella che sto usando?
- Come posso ottenere in modo sicuro la chiave pubblica dell'utente con cui voglio comunicare (con che certezza posso dire che quella chiave pubblica appartenga effettivamente a quell'utente)?

> Il certificato digitale, emesso da una Certification Authority (CA), certifica il legame utente/chiave pubblica (il tutto avviene in quello che viene chiamata **Public Key Infrastructure**):
>
> - L'utente deve essere ricondotto ad un soggetto (persona, computer, organizzazione);
> - Il certificato e' firmato digitalmente da una CA che garantisce l'associazione;
> - Il formato tipico e' lo standard X.509.

Certification Authority

E' un'entita' che emette certificati firmati con la sua chiave privata. Quest'entita' ha una chiave pubblica che deve essere nota, per ovviare al problema della distribuzione sicura della sua chiave ed emette i certificati su **richiesta dell'utente**.

La CA, in fase di emissione di un certificato, verifica che:

- L'utente sia identificabile chiaramente con il nome dichiarato (esiste un nome simile/problemi di omonimia);
- L'utente sia in possesso della chiave privata corrispondente a quella pubblica per cui chiede il certificato.

Rilascio di certificati

Alice vuole generare una coppia di chiavi (K_A^+, K_A^-) (rispettivamente pubblica/privata) e vuole rendere K_A^+ pubblica:

1. Alice spedisce in modo sicuro K_A^+ alla CA;
2. Alice prova la sua identita' alla CA;
3. La CA verifica che Alice conosca K_A^-;
4. La CA genera il certificato C_{K_A} che contiene la **chiave pubblica e i dati identificativi di Alice**;

- La CA firma il certificato con la propria chiave privata e lo spedisce ad Alice.

I punti 1 e 2 sono gli anelli deboli della catena in quanto se non implementati correttamente permettono a Trudy di fare in modo che la CA emetta un certificato digitale per Alice, contenente pero' la chiave pubblica di Trudy, il che le consente di impersonificare Alice.

Utilizzo di certificati

Quando Alice comunica con Bob:

- Allega sempre il certificato, oppure lo spedisce a richiesta:
  - La CA non deve essere sempre online (come nel caso dei Key Distribution Center);
- Il certificato di Alice puo' venire verificato da coloro che conoscono la chiave pubblica della CA.

La CA diventa il singolo punto di fallimento e la vulnerabilita' aumenta col numero di partecipanti. Inoltre, la CA deve rendere pubblica in modo sicuro la propria chiave pubblica.

PKI - Public Key Infrastructure

I problemi che abbiamo rilevato vengono ovviati dal fatto che esiste una gerarchia di CA:

- CA1 puo' delegare CA2 nell'emissione di certificati;
- LA delega e la gerarchia tra CA e' espressa mediante una catena di certificati;
- La **Root Authority** e' la CA di livello piu' alto:
  - Usa un certificato **self-signed**, in genere memorizzato dal produttore nel browser, risolvendo il problema della distribuzione della chiave pubblica;
  - Le CA subordinate hanno certificati firmati dalla Root Authority, mentre gli altri hanno certificati firmati dalla CA di livello superiore.

Catene di certificati

Nella PKI abbiamo certificati di utenti e certificati di CA, in particolare abbiamo **catene di certificati** che rappresentano una serie di certificati emessi da CA successive e tracciano un **percorso** nella gerarchia, che va dalla Root Authority al certificato emesso per uno specifico utente.

Ogni certificato e' firmato dalla chiave privata del suo emettitore e la firma puo' venire verificata con la chiave pubblica dell'emettitore che e' contenuta nel certificato successivo della catena.

Quando viene presentato un certificato, esso viene verificato secondo tramite alcuni passaggi:

- Viene verificato il periodo di validita' del certificato;
- Viene localizzato il certificato dell'emettitore (che puo' trovarsi del database locale oppure nella catena di certificati fornita dal soggetto che ha presentato il certificato);
- La firma del certificato viene verificata usando la chiave pubblica contenuta nel certificato dell'emettitore;
- Se il certificato dell'emettitore e' fidato (in quanto e' presente nel database locale) la verifica si ferma, altrimenti:
  - Si ricomincia la valutazione per il certificato successivo della catena.

Formato dei certificati

Lo standard X.509 richiede che il certificato contenga:

- I dati identificativi del proprietario della chiave pubblica, che possono riferirsi a una persona, un computer o un'organizzazione;
- La chiave pubblica;
- Il periodo di validita' del certificato;
- Un link alla lista dei **certificati revocati** (CRL);
- La firma della CA che ha emesso il certificato.

Revoca del certificato

Un utente o un emettitore possono richiedere la revoca di un certificato prima della data di scadenza di validita' del certificato (per cambio dati personali, licenziamento, compromissione chiave privata).

La revoca avviene tramite la generazione di una **CRL - Certificate Revocation List** che contiene tutti i certificati revocati prima della loro scadenza naturale. La lista e' firmata dalla CA che ha emesso il certificato ora revocato.

KDC vs PKI

KDC - crittografia simmetrica

- Il KDC deve essere sempre online in quanto usato ad ogni sessione;
- Il KDC conosce la **chiave segreta**;
- Se il KDC viene compromesso, vengono esposti i messaggi passati e futuri;
- Veloce.

PKI - crittografia asimmetrica

- La CA puo' essere offline tranne che in fase di generazione di certificati;
- La CA conosce solo le **chiavi pubbliche**;
- Se la CA viene compromessa, vengono esposti solo i messaggi futuri;
- Lento.

Protocolli crittografici

Notazione e caratteristiche di base

Obiettivi

I protocolli possono avere come obiettivo quello di garantire una o piu' proprieta' di sicurezza, in particolare:

- **Autenticazione** (unilaterale o mutua): sia delle parti che del messaggio;
- **Scambio di chiavi (simmetriche) di sessione**: chiave segreta condivisa richiesta dagli algoritmi di cifratura simmetrici;
- **Scambio di dati (garantendo integrita' e confidenzialita')**: scambio di informazioni confidenziali mediante un canale insicuro;
- Una combinazione degli obiettivi.

Terminologia

- **Sessione (o run) del protocollo**: singola esecuzione del protocollo dall'inizio alla fine;
- **Chiavi a lungo/breve termine**:
  - Lungo termine: sono chiavi che esistono prima che il protocollo inizi;
  - Breve termine (**chiavi di sessione**): sono chiavi strettamente legate ad una singola esecuzione del protocollo:
    - Si puo' rendere pubblica la chiave alla fine della sessione del protocollo;
    - Puo' essere debole dal punto di vista crittografico (perche' se anche venisse scoperta tramite un bruteforce, la chiave ormai non sara' piu' valida perche' scaduta).

Caratteristiche

- L'autenticazione puo' essere richiesta o meno:
  - Nel caso in cui fosse richiesta bisogna specificare se unilaterale o mutua;
- Le chiavi vengono cambiate frequentemente (freshness);
- Efficienza del protocollo;
- Presenza richiesta o meno di un terzo partecipante, fidato o meno;
- Controllo delle chiavi;
- Richiesto di garantire anche la non-repudiation.

Quando si analizza un protocollo crittografico per vedere se puo' subire degli attacchi si deve considerare che:

- Ci possono essere molte esecuzioni dello stesso o diversi protocolli in contemporanea;
- Lo **stesso partecipante** puo' prendere parte a diverse esecuzioni dello stesso o diversi protocolli, magari anche in ruoli diversi;
- I partecipanti possono essre sia online, sia offline durante l'esecuzione del protocollo;
- **Non tutti i partecipanti possono essere del tutto fidati**.

Entita'

- **Partecipanti generici**: indicati con le prime lettere dell'alfabeto:
  - A (Alice);
  - B (Bob);
  - C (Charlie);
- **Attaccanti**:
  - T (Trudy, intruder);
  - E (Eve)
    - E(A) indica Eve che finge di essere Alice;
  - M (Mallory, malicious user);
- **Server**
  - S (Sam);
  - Nomi piu' specifici:
    - CA (Certification Authority);
    - KDC (Key Distribution Center);
    - TTP (Trusted Third Party).

Chiavi, cifrature, firme e certificati digitali

- **Chiavi**:
  - K: chiave **simmetrica**;
  - K_{AB}: chiave **simmetrica** condivisa da Alice e Bob;
  - K_A o K_A^+: chiave **pubblica** di Alice;
  - K_A^{-1} o K_A^-: chiave **privata** di Alice;
- **Cifrature**:
  - $\{m\}_K$: cifratura di m con la chiave K;
- **Firme**:
  - $[m]_{K^-}$ o $\{m\}_{K^-}$: firma di m con chiave K^-;
- **Certificati digitali**:
  - **CA**: certificato di Alice;
  - **CB**: certificato di Bob;

Notazione

Un protocollo specifica l'ordine e il formato dei messaggi trasmessi: dunque per esprimere un protocollo si possono elencare i messaggi scambiati oppure si puo utilizzare una notazione grafica.

- Elenco:
  - 4. B -> A: $\{T_A + 1\}_{K_{AB}}$
    - Numero del messaggio;
    - Mittente e destinatario;
    - Messaggio:
      - Contenuto:
        - Timestamp a cui e' stato sommato il valore 1;
      - Cifrato con la chiave condivisa K_{AB};
- Notazione grafica mediante **Sequence diagram**

Esempio di protocollo di autenticazione

Progettazione di un semplice protocollo di autenticazione unilaterale:

Obiettivo 1: Bob vuole che Alice dimostri la sua identita'.

- Protocollo 1:
  - *1. A->B: {"Sono Alice"}*
    - Questo protocollo non funziona in quanto Alice, usando un canale insicuro, puo' incappare in Trudy che vuole attaccare il normale funzionamento del protocollo. Poiche' Trudy puo' essere un attaccante attivo e quindi immettere nuovi messaggi lungo il canale, puo' dichiarare di essere Alice: *1. T(A)->B: {"Sono Alice"}*. Bob puo' non accorgersi di questa cosa in quanto non puo' vedere Alice.
- Protocollo 2:
  - *2. A->B: {"Sono Alice", IPAddress}*
    - Trudy puo' creare un messaggio recuperando l'IPAddress di Alice mediante **spoofing** e quindi impersonificare Alice.
- Protocollo 3:
  - *3. A->B: {"Sono Alice", password}*
    - Anche inviando come messaggio una password nota solo ad Alice e Bob, Trudy puo' effettuare un **replay attack** memorizzando il messaggio di Alice per riutilizzarlo in seguito con Bob.
- Protocollo 4:
  - *4. A->B: {"Sono Alice", encrypt(password)}*
    - Il protocollo e' piu' sicuro, ma Trudy puo' effettuare di nuovo un **replay attack**, la differenza sta nel fatto che ora Trudy non conosce la password in chiaro di Alice.

Obiettivo 2: Fare in modo che Alice dimostri la sua identita' e non ci siano replay attack.

- Protocollo 5:
  - *5. A->B: {"Sono Alice"}*;
  - *5' B->A: {nonce N}* (il **nonce** e' un numero casuale usato solo una volta);
  - *5'' A->B:* $\{\{N\}_{K_{AB}}\}$
    - In questo caso il protocollo funziona, il problema risiede nel fatto che Alice e Bob devono condividere la chiave segreta K_{AB}, quindi non hanno modo di scambiarsi le chiavi in modo sicuro prima di eseguire il protocollo.
- Protocollo 6:
  - *6. A->B: {"Sono Alice"}*;
  - *6' B->A: {nonce N}*;
  - *6'' A->B:* $\{\{N\}_{K_A^-}\}$;
  - *6''' B->A: {"Spediscimi la chiave pubblica"}*;
  - *6'''' A->B:* $\{K_A^+\}$;
  - Bob verifica la firma di Alice calcolando $K_A^+(K_A^-(N)) = N$.
    - Questo protocollo non puo' subire un attacco di tipo replay, puo' subire un **man in the middle** in cui Trudy finge di essere Alice con Bob e Bob con Alice: questa cosa e' possibile in quanto Bob non e' a conoscenza della chiave pubblica di Alice e quindi la chiede a quella che dice di essere Alice (in realta' Trudy) che invia la propria chiave pubblica. **Trudy riesce a leggere tutti i messaggi** perche' Bob cifrera' tutti i messaggi con la chiave pubblica di Trudy.
    - La soluzione risiede nell'utilizzo di **chiavi pubbliche certificate**, quindi Alice non dovra' spedire la propria chiave pubblica in chiaro, ma inviera' un certificato firmato da una CA.

Attacchi comuni

- **Osservazione passiva dei messaggi**
  - Contromisura: uso della crittografia per garantire segretezza;
- **Intercettazione e modifica dei messaggi**
  - Contromisura: uso di funzioni hash o primitive di cifratura per legare insieme le informazioni o per fare in modo che solo chi possiede una certa chiave possa generare i messaggi;
- **Replay attack**: utilizzo di messaggi vecchi
  - Contromisura: specificare il contesto in modo che un messaggio non possa venire utilizzato piu' volte in una stessa esecuzione o in esecuzioni diverse del protocollo;
- **Attacco alla freshness**: utilizzo di parte di vecchi messaggi (chiave, nonce, ecc)
  - Contromisura: usare meccanismi per fare in modo che un partecipante legittimo al protocollo risconosca se l'informazione e' fresh o meno;

- **Attacco a sessioni parallele**: l'attaccante combina i messaggi di sessioni diverse
- **Reflection attack**: un tipo di replay attack dove l'attaccante utilizza il protocollo contro se stesso. Rispedisce l'informazione o il messaggio al mittente che l'ha spedito, facendo in modo di ottenere da lui il messaggio che deve spedirgli nella sessione del protocollo in cui i ruoli sono invertiti
- **Men in the middle**: l'attaccante crea delle connessioni indipendenti con i 2 partecipanti e li impersona (entrambi o soltanto uno). Si mette dunque in mezzo alla normale esecuzione del protocollo facendo uso di sessioni parallele. Questo attacco e' sempre possibile, ma ha senso solo se l'attaccante ne trae vantaggio.
- **Denial of service**: ogni richiesta di comunicazione a un server utilizza una certa quantita' di memoria e CPU, quindi l'attaccante cerca di far utilizzare tutta la memoria o la CPU del server facendo in breve tempo un elevato numero di richieste. Tutti i sistemi possono subire un tale attacco, ma ci sono alcuni protocolli che si comportano meglio.

Denial of Service

Un classico attacco DoS e' quello denominato **SYN flood DoS Attack** alla fase di hadshake del protocollo TCP. In genere quando un client e un server entrano in comunicazione per la prima volta scambiano 3 messaggi per effettuare l'handshake:

- 1. A->S: {SYN}
  - Il client invia al server un messaggio di sincronizzazione;
- 2. S->A: {ACK, SYN}
  - Il server risponde al client con un messaggio di Acknowledgment di richiesta del client e una richiesta di sincronizzazione con il client;
  - Il server aggiunge il client A alla tabella delle connessioni;
  - Il server rimane in attesa di risposta di tipo Acknowledgement da parte del client (timeout in 3 min);
- 3. A->S: {ACK}

L'attacco SYN flood invia moltissimi messaggi di richiesta di sincronizzazione (SYN) al server, tale che questo va a riempiere le sue tabelle (flood), per cui una successiva richiesta (legittima o meno) viene ignorata perche il server non ha piu' memoria a sufficienza.

Protocolli challenge-response

Descrizione

Il protocollo challenge-response e' un protocollo di autenticazione, per questo definiti anche **test di autenticazione**. Si tratta di protocolli minimali per ottenere unilaterale o mutua autenticazione. In genere rappresenta la parte di setup di un protocollo piu' grande.

Si basano su un meccanismo particolare in cui un partecipante prova la sua identita' dimostrando di conoscere un segreto (non necessariamente condiviso) e un'informazione che varia nel tempo (per evitare replay attacks) senza rivelare esplicitamente il segreto.

Funzionamento base (con chiave condivisa)

In genere questo protocollo richiede lo scambio di due messaggi:

- **Challenge**: il sistema o l'altro partecipante presenta una stringa contenente l'informazione che varia nel tempo;
- **Response**: l'utente cifra la stringa con la chiave condivisa $\{string\}_{K_{AB}}$ e la spedisce all'altro partecipante;
- **Autenticazione**: il partecipante controlla la stringa $\{string\}_{K_{AB}}$ in quanto e' in grado di decifrarla perche' conosce la chiave. Se la stringa coincide con quella spedita inizialmente allora l'utente viene autenticato.

Perche' il protocollo funzioni correttamente sia il challenge che il response dovrebbero essere *fresh*, in genere puo' essere:

- Nonce;
- Timestamp;
- Numeri in sequenza;
- Chiave a breve termine;

Mantenere la freshness

Nonce

Il nonce e' una sequenza casuale di bit (32-128), generata ogni volta dal mittente come challenge, quindi non prevedibile e verificata solo nel response (memorizzata solo dal mittente). Il **destinatario non la controlla**, perche' sarebbe costoso e poco pratico. L'importante e' che non venga **mai riutilizzata**, altrimenti non sarebbe garantita la freshness.

- Esempio 1:
  - 1. A->B: {N}
  - 2. B->A: $\{N\}_{K_{AB}}$
- Esempio 2:
  - 1. A->B: $\{N\}_{K_{AB}}$
  - 2. B->A: {N}
- Esempio 3:
  - 1. A->B: $\{N\}_{K_{AB}}$
  - 2. B->A: $\{N-1\}_{K_{AB}}$

Timestamp

Rappresenta l'ora nel computer locale in millisecondi. E' un'informazione **controllabile dal destinatario** e **richiede sincronizzazione**. Poiche' c'e' sincronizzazione tra i due e' un'informazione prevedibile: per questo motivo il destinatario deve memorizzare i timestamp recenti per evitare riutilizzi. Dunque non sono possibili tutte le configurazioni del protocollo come per il nonce (perche se Trudy riuscisse a sincronizzarsi con Alice e Bob sarebbe in grado di inviare a nome di Bob il timestamp in chiaro nella response):

- Esempio 1:
  - 1. A->B: {t}
  - 2. B->A: $\{t\}_{K_{AB}}$
- Esempio 2:
  - 1. A->B: $\{t\}_{K_{AB}}$
  - 2. B->A: $\{t-1\}_{K_{AB}}$

Numeri in sequenza

Il mittente mantiene un **contatore** *c* che viene incrementato di 1 dopo ogni challenge e deve essere legato ai dati che identificano il canale e la coppia di partecipanti. Ci sara' un **identificatore specifico per sessione/coppia di partecipanti**.

Il destinatario memorizza il valore piu' recente e non accetta valori troppo vecchi. E' un meccanismo simile al timestamp, ma e' locale al mittente e non richiede sincronizzazione.

- Esempio:
  - 1. A->B: {c}
  - 2. B->A: $\{c\}_{K_{AB}}$

Chiave a breve termine

Il mittente genera una chiave *J* e la spedisce cifrata. Il destinatario invia un messaggio qualsiasi usando *J* per la cifratura dimostrando di essere stato in grado di decifrare il messaggio (quindi di conoscere l'informazione segreta condivisa):

- 1. A->B: $\{J\}_{K_{AB}}$
- 2. B->A: $\{\text{"Ciao"}\}_{J}$

In questo modo si ottiene la distribuzione della chiave di sessione oltre all'autenticazione e quindi e' un meccanismo piuttosto utile.

Spesso si utilizza un **terzo partecipante** per fargli generare una chiave a breve termine. Alice invece di inviare il challenge a Bob, invia la chiave di sessione e l'identita' di Bob al Server (col quale deve condividere una chiave segreta). Il Server invia la chiave di sessione a Bob (cifrato con la chiave condivisa tra Bob e il Server). Bob si autentica con Alice inviando il response cifrato la chiave di sessione inviata all'inizio da Alice, dimostrando di essere stato in grado di decifrare l'informazione e riutilizzare la chiave di sessione proposta:

- 1. A->S: $\{J\}_{K_{AS}}$
- 2. S->B: $\{J\}_{K_{BS}}$
- 3. B->A: $\{"Ciao"\}_J$

Protocollo di Needham-Schroeder a chiave condivisa

Fu inventato nel 1978 da Needham e Schroeder.

Descrizione

Si tratta di un protocollo a chiave segreta per stabilire una chiave di sessione utilizzabile da due entita' di rete per proteggere le comunicazioni successive.

- Fa uso della crittografia simmetrica;
- Usa un KDC per generare la chiave di sessione;
- Richiede l'autenticazione mutua tramite challenge-response e l'invio di nonce;
- Utilizza 2 chiavi condivise:
  - Una a lungo termine condivisa con il server;
  - Una a di sessione generata dal server.

Funzionamento

Esempio:

- 1. A->S: $\{A, B, N_A\}$
  - Alice invia al Server l'identita' di Alice, Bob e invia un nonce da lei generato.
- 2. S->A: $\{n_a, B, K_{AB}, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$
  - Il server genera la chiave di sessione K_{AB};
  - Risponde ad Alice inviando:
    - La chiave di sessione K_{AB} appena generata;
    - La coppia K_{AB}, identita' di A crittata con la chiave K_{BS} in modo che possa essere inoltrata a Bob per renderlo partecipe al protocollo;
    - Il nonce n_a e l'identificativo di Bob, perche' Alice puo' inviare piu' richieste al Server per iniziare piu' comunicazioni e deve quindi essere in grado di distinguere le varie risposte del Server.
  - Il messaggio inviato ad Alice viene crittato con la chiave K_{AS} nota solo ad Alice e al Server.
- 3. A->B: $\{K_{AB}, A\}_{K_{BS}}$
  - Alice invia a Bob la chiave di sessione K_{AB} e il proprio identificativo, criptati con la chiave K_{BS} ricevuta dal Server;
  - Bob e' l'unico (oltre al Server) in grado di decriptare questo messaggio.
- 4. B->A: $\{n_b\}_{K_{AB}}$
  - Una volta decriptato il messaggio, Bob invia ad Alice un nonce (come challenge per fare in modo che A si autentichi con lui) criptato con la chiave di sessione K_{AB} per mostrare che e' in possesso della chiave, autendicandosi con A.
- 5. A->B: $\{n_b\text{-}1\}_{K_{AB}}$
  - Alice decifra il nonce di Bob, lo modifica, lo cifra con la chiave di sessione e lo rispedisce indietro autenticandosi con Bob.

In questo cambio di messaggi, l'attaccante e' sempre presente e puo' leggere tutti i messaggi trasmessi lungo il canale, sia quelli tra Alice e il Server che quelli tra Alice e Bob.

Possibili attacchi

Nel 1981 fu trovato un possibile attacco da Denning e Sacco tramite un replay attack che utilizza una **vecchia chiave compromessa** K_{AB}.

Dopo lo scambio dei 5 messaggi tra Alice e Bob in cui condividono la chiave di sessione K_{AB} e si autenticano tra di loro, Trudy (con tutto il tempo necessario) effettua un attacco bruteforce sul messaggio numero 3.

Aquesto punto Trudy rispedisce a Bob il messaggio 3 di Alice, e Bob lo accetta visto che non riesce a capire che si tratta di una vecchia chiave di sessione:

- 3!. T->B: $\{K_{AB}, A\}_{K_{BS}}$
- 4!. B->T: $\{n'_b\}_{K_{AB}}$
  - Bob non accorgendosi che si tratta di una vecchia chiave di sessione, risponde a Trudy, credendo sia Alice.
- 5!. T->B: $\{n'_b\text{-}1\}_{K_{AB}}$
  - Trudy e' ora in grado di decifrare tutti i messaggi e quindi l'attacco ha successo.

Contromisure

Una possibile soluzione e' quella di inserire un timestamp o un nonce nel messaggio numero 2, creato dal Server e inviato ad Alice, ma destinato a Bob: $\{K_{AB}, A\}_{K_{BS}}\}$.

Il protocollo Kerberos fa uso del timestamp per risolvere il problema della vecchia chiave compromessa:

- 3*. A->B: $\{K_{AB}, A, t_{exp}\}_{K_{BS}}$
- 4*. B->A: $\{n_b\}_{K_{AB}}$
- 5*. A->B: $\{n_b\text{-}1\}_{K_{AB}}$

In questo caso l'attacco non e' piu' possibile perche' quando Bob riceve il messaggio verifica che il timestamp sia ancora valido, ovvero che la chiave sia ancora fresh.

Protocollo di Needham-Schroeder a chiave pubblica

Anche questo protocollo fu inventato nel 1978 da Needham e Schroeder.

Descrizione

Questo protocollo ha come obiettivo la mutua autenticazione tra due entita' di rete.

Caratteristiche:

- Utilizzo della crittografia asimmetrica;
- Utilizzo di una CA per certificare le chiavi pubbliche;
- Mutua autenticazione tramite challenge-response basata sul nonce;
- Al termine dell'esecuzione del protocollo A e B condividono 2 numeri segreti non noti a chiunque non conosca K_A^- e K_B^- (questi numeri segreti potrebbero essere utilizzati come chiave a breve termine nella comunicazione delle due direzioni).

Funzionamento

- 1. A->S: {A,B}
  - Alice chiede al Server la chiave pubblica di Bob K_B^+ inviando la propria identita' e quella di Bob.
- 2. S->A: $[B,K_B^+]_{K_S^-}$
  - Il Server risponde ad Alice inviando il certificato di Bob $[B,K_B^+]$ firmati con la chiave privata del Server K_S^-;
  - Alice puo' usare la chiave pubblica del Server K_S^+ per verificare la firma del Server e verificare che il certificato contenga l'identificativo di Bob.
- 3. A->B: $\{A, n_A\}_{K_B^+}$
  - Alice invia un challenge a Bob generando un nonce n_A e cifrandolo con la chiave appena ricevuta, ovvero la chiave pubblica di Bob K_B^+.
- 4. B->S: {B, A}

    - Bob decifra il messaggio ricevuto da Alice con la sua chiave privata K_B^- e chiede la chiave pubblica di Alice al Server.
- 5. S->B: $[A, K_A^+]_{K_S^-}$
    - Il Server soddisfa la richiesta di Bob inviando il Certificato di Alice a Bob;
    - Bob verifica la firma del Server e recupera la chiave pubblica di Alice.
- 6. B->A: $\{n_A, n_B\}_{K_A^+}$
    - Bob genera un nonce n_B e lo invia ad Alice insieme al nonce n_A che Alice gli aveva mandato in precedenza, crittando il messaggio con la chiave pubblica di Alice K_A^+;
    - In questo modo Bob ha inviato la response al challenge di Alice dimostrando che e' in possesso della chiave privata K_B^-.
- 7. A->B: $\{n_B\}_{K_B^+}$
    - Alice invia il response a Bob, cifrato con la chiave pubblica di Bob K_B^+ provando che e' in possesso della chiave privata K_A^-.

In questo modo Alice e Bob si sono autenticati e sono gli unici a conoscere i nonce n_A e n_B.

Possibili attacchi

Dopo 10 anni dalla pubblicazione del protocollo, Lowe ha scoperto una vulnerabilita' non voluta del protocollo che si basa su due possibili scenari di attacco:

- **Dall'interno**:
    - Alice vuole veramente parlare con Trudy, non aspettandosi che sia un utente malevolo, che apre una sessione parallela con Bob fingendosi Alice;
- **Dall'esterno**:
    - Alice vuole parlare con Bob, ma si assume che Trudy sia riuscita a sostituire la chiave pubblica di Bob con la propria e quindi Trudy si finge Alice con Bob e Bob con Alice.

In dettaglio:

- 1. A->T: $\{A, n_A\}_{K_T^+}$
- 2. T->B: $\{A, n_A\}_{K_B^+}$
    - Trudy, dopo essersi messa con contatto con Alice, apre una sessione parallela con Bob, inoltrando i messaggi Alice.
- 3. B->A: $\{n_A, n_B\}_{K_A^+}$
    - Bob manda ad Alice i due nonce n_A e n_B, ma Alice e' convinta di riceverlo da Trudy.
- 4. A->T: $\{n_B\}_{K_T^+}$
    - Alice, convinta che il messaggio precedente sia stato inviato da Trudy, quindi le invia il nonce n_B cifrato con la chiave pubblica di Trudy K_T^+;
    - Trudy e' in grado di decifrare il messaggio ricevuto.
- 5. T->B: $\{n_B\}_{K_B^+}$
    - Trudy cifra il nonce con la chiave pubblica di Bob K_B^+ facendo in modo che Bob sia convinto di essere in comunicazione con Alice.

Contromisure

Per proteggersi da questa vulnerabilita' Lowe propone di aggiungere l'identita' del mittente del messaggio nel messaggio contenente i 2 nonce.

L'attacco man in the middle in questa modalita' non e' piu' possibile, in quanto, quando Alice riceve il messaggio coi 2 nonce si accorge che il messaggio e' stato inviato da Bob e non da Trudy.

Trudy potrebbe fare in modo che il messaggio non arrivi ad Alice, ma se bloccasse il messaggio non avrebbe la possibilita' di conoscere il nonce n_B e quindi di inviare il nonce corretto che Bob si aspetta di ricevere da Alice.

Proprieta'

Con questi accorgimenti, **il ricevente viene correttamente autenticato**: se il mittente Alice completa il protocollo crede che Bob sia il ricevente, questo implica che **Bob ha risposto a chi pensa essere Alice**. L'autenticazione unilaterale in questo senso funziona.

Anche **il mittente viene correttamente autenticato** in quanto se il ricevente Bob completa il protocollo, crede che Alice sia il mittente e quindi **Alice pensa di aver iniziato il protocollo con Bob**. Anche in questo caso l'autenticazione unilaterale si conclude correttamente.

E' garantita la **fresheness e la segretezza dei nonce** perche' se un mittente onesto completa il protocollo con un ricevente onesto, l'attaccante non conosce i nonce.

Principi di buona progettazione

Abadi e Needham nel 1996 hanno proposto una serie di principi di buona progettazione che aiutano a prevenire gli attacchi. Questi principi pero' non garantiscono la correttezza del protocollo, in quanto non sono condizioni necessarie ne sufficienti per poter affermare che il protocollo sia sicuro. Quindi vanno utilizzati insieme ad altri metodi come la **verifica formale**.

Principio 0

Il protocollo **deve essere efficiente**:

- Cifrare solo se necessario;
- Non includere parti di messaggio di cui non c'e' bisogno.

Il principio 0 e' in conflitto con gran parte degli altri principi, quindi bisogna di volta in volta decidere quale principio ha maggior peso.

Un protocollo che non soddisfa il principio 0 e' la **versione II di Kerberos**:

- 1. A->S: $\{A, B, n_A\}$
- 2. S->A: $\{K_{AB}, B, L, n_A, \{K_{AB}, A, L\}_{K_{BS}}\}_{K_{AS}}$
- 3. A->B: $\{A, T_A\}_{K_{AB}}, \{K_{AB}, A, L\}_{K_{BS}}$
- 4. B->A: $\{T_A+1\}_{K_{AB}}$

Alice, tramite un Server che funge da KDC, concorda una chiave di sessione K_{AB} con Bob.

Nel secondo messaggio avviene una doppia cifrattura non necessaria (fortunatamente si tratta di crittografia simmetrica quindi non e' troppo inefficiente). Questo problema esiste anche nel protocollo Needham-Schroeder a chiave condivisa.

L'elemento *L* (lifetime) nei messaggi e' la durata di vita della chiave di sessione che risolve il problema sollevato da Denning e Sacco.

Per ottimizzare questo protocollo rimuoviamo la doppia cifratura:

- 1. A->S: $\{A, B, n_A\}$
- 2. S->A: $\{K_{AB}, B, L, n_A\}_{K_{AS}}, \{K_{AB}, A, L\}_{K_{BS}}$
- 3. A->B: $\{A, T_A\}_{K_{AB}}, \{K_{AB}, A, L\}_{K_{BS}}$
- 4. B->A: $\{T_A+1\}_{K_{AB}}$

Principio 3

> Nel caso in cui l'identita' dei partecipanti sia essenziale per comprendere il significato di un messaggio, e' necessario menzionare anche il **nome dei partecipanti** al protocollo all'interno del messaggio.

Nel **protocollo di Denning e Sacco** per stabilire una chiave di sessione, questa proprieta' non e' soddisfatta, infatti il protocollo e' vulnerabile ad un attacco. Il protocollo usa la crittografia a chiave pubblica e si appoggia a delle CA per ottenere i certificati e quindi le chiavi pubbliche.

- 1. A->S: $\{A, B\}$
- 2. S->A: $\{C_A, C_B\}$
- 3. A->B: $\{C_A, C_B, \{\{K_{AB}, T_A\}_{K_A^-}\}_{K_B^+}\}$
  - La chiave di sessione K_{AB} viene firmata da Alice con la sua chiave privata K_A^- e poi la cifra con la chiave pubblica di Bob K_B^+ in modo da garantire autenticazione, integrita' e confidenzialita' della chiave;
  - Alice mette anche un timestamp T_A per stabilire una durata di vita della chiave.

Problema

Il problema e' che Bob per tutta la durata di T_A puo' pretendere di essere Alice con una terza persona: decifrando il messaggio e presentando la parte di messaggio firmata da Alice.

Soluzione

La soluzione a questo problema e' inserire l'identita' di Alice e Bob all'interno del messaggio firmato da Alice, in modo che chi verifica la firma si accorga per chi era intesa quella parte di messaggio:

- 3. A->B: $\{C_A, C_B, \{\{A, B, K_{AB}, T_A\}_{K_A^-}\}_{K_B^+}\}$

Principio 4

> Bisogna specificare chiaramente i motivi per cui si cifrano parti di messaggio. Cifrare non e' sinonimo di sicurezza e un uso improprio puo' causare problemi.

La cifratura viene utilizzata per:

- Ottenere confidenzialita':
  - Si assume che solo le persone autorizzate posseggano la chiave da utilizzare per decifrare il messaggio;
- Garantire autenticazione:
  - Solo il mittente legittimo conosce la chiave di cifratura;
- Legare insieme parti di messaggio:
  - Ricevere $\{X, Y\}_K$ non e' sempre equivalente a ricevere $\{X\}_K$ e $\{Y\}_K$;
- Produrre numeri casuali.

Principio 5

> Non e' detto che la persona che firma un messaggio che e' stato precedentemente cifrato ne conoscesse il contenuto. Che e' duale a: Non e' detto che la persona che firma un messaggio che poi viene cifrato intendesse spedirlo proprio alla persona che presenta la firma cifrata.

Quindi bisogna decidere se e' meglio firmare prima di cifrare o il contrario.

Lo standard **CCITT X.509** contiene 3 diversi protocolli per la comunicazione sicura e firmata che assume che i partecipanti conoscano le chiavi pubbliche e non debbano ricorrere ad una CA per ottenerle:

- 1. A->B: $\{A, \{T_A, N_A, B, X_A, \{Y_A\}_{K_B}\}_{K_A^-}\}$
  - T_A e' il timestamp;
  - N_A e' un nonce non utilizzato;
  - X_A e Y_A sono dei dati dell'utente

Il protocollo dovrebbe garantire l'integrita' di X_A e Y_A, la loro origine e la segretezza di Y_A. Bob pero' non e' siocuro che Alice conosca Y_A quindi la proprieta' di non ripudio nono e' garantita.

Attacco

E' possibile un attacco in cui una terza parte intercetta il messaggio, rimuove la firma e mette la propria copiando la parte di messaggio cifrato all'iterno della firma senza comprenderne il contenuto:

- 1. A->E(B): $\{A, \{T_A, N_A, B, X, \{Y\}_{K_B}\}_{K_A^-}\}$
- 2!. E->B: $\{E, \{T_A, N_A, B, X, \{Y\}_{K_B}\}_{K_E^-}\}$

Sembrerebbe sia meglio **firmare e poi cifrare**. Ma per il duale del principio anche firmare e poi cifrare puo' causare un problema.

Riprendendo il protocollo di Denning e Sacco:

- 1. A->S: $\{A, B\}$
- 2. S->A: $\{C_A, C_B\}$

- 3. A->B: $\{C_A, C_B, \{\{K_{AB}, T_A\}_{K_A^-}\}_{K_B^+}\}$

Nel terzo messaggio Alice genera una chiave di sessione K_{AB} condivisa con Alice e Bob, la firma e poi la cifra per garantirne la confidenzialita'.

Bob pero' per tutta la durata del timestamp T_A puo' impersonare Alice sfruttando il fatto che puo' recuperare un messaggio firmato da lei:

- 4. B(A)->D: $\{C_A, C_B, \{\{K_{AB}, T_A\}_{K_A^-}\}_{K_D^+}\}$

Piu' in generale va bene **cifrare prima di firmare** quando la non-repudiation non e' di primaria importanza. Negli altri casi e' importante esaminare attentamente il protocollo e decidere caso per caso.