Appunti - Sicurezza delle reti

Martini Michele (e tante altre bestie)

27 gennaio 2020

1. Sicurezza informatica

Cosa si intende con *information security*?

- **Computer security**: prevenzione e rilevazione di azioni non autorizzate da parte di utenti di un singolo computer.
- **Network security**: politiche e sistemi adottati dall'amministratore per proteggere la rete e le risorse in essa contenute da accessi non autorizzati.
- **Information security**: sicurezza in senso generico, indica la protezione di informazioni e information systems da accessi/manipolazioni non autorizzati.

1.1 Sicurezza come policy compliance

Adottando una prospettiva ben nota in software engineering, possiamo definire i meccanismi di sicurezza per sistemi e risorse in base alla conformità ad una policy o a goals designati:

- *Policy:* Specifica dei comportamenti permessi o meno nel sistema.
- *Meccanismo:* Meccanismo implementato per imporre la policy.
- *Compliance:* Conformità alle specifiche di sicurezza.

Al fine di evitare fraintendimenti, diamo una definizione più precisa rigurdo i termini: proprietà, policy e meccanismo.
Una **proprietà** è una descrizione ad alto livello di ciò che si desidera ottenere.
Un **meccanismo** è l'implementazione pratica dei sistemi che permettono l'attuazione di una proprietà.
Una **policy** è la definizione precisa e disambigua di una proprietà, che permette di distinguere un comportamento regolare da uno anomalo. Essa consta di tre parti: prevenzione, detection e risposta.

Tradizionali proprietà di sicurezza

Introduciamo e studiamo le più comuni proprietà di sicurezza:

- *Confidentiality*
  L'informazione non dev'essere ottenibile/comprensibile da utenti non autorizzati.
- *Integrity*
  L'informazione non dev'essere modificabile da utenti non autorizzati.
- *Availability*
  L'informazione dev'essere sempre disponibile.

- *Accountability*
  Le azioni eseguite devono essere sempre tracciabili.
- *Authentication*
  I principals e l'origine dei dati devono sempre essere identificabili.

Confidenzialità, privacy, segretezza Dati e informazioni memorizzate in un sistema o scambiate tra due entità devono essere protette da accessi non autorizzati, ovvero devono risultare accessibili solo agli utenti e ai processi che ne hanno diritto, in base alle policy definite nel sistema. Solitamente se la policy riguarda un singolo individuo parliamo di *privacy*, nel caso di un'organizzazione parliamo invece di *secrecy*. La confidenzialità si ottiene principalmente mediante tecniche crittografiche.
NB: privacy $\neq$ anonimity. *Anonimity*: l'identità del principal è celata.

Integrity L'integrità garantisce che i dati non vengano alterati da principal malevoli. Essa presuppone una security polity atta ad identificare gli utenti autorizzati alla modifica dei dati (quali utenti possono manipolare quali dati).

Disponibilità Questa proprietà assicura l'accesso a dati e servizi in modo affidabile ed in tempi congrui, proteggendo le risorse da: attacchi malevoli, attacchi accidentali (utenti distratti) ed eventi ambientali (es: incendi).
Non è sempre facile distinguere un possibile attacco (es: attacco DOS) rispetto ad un uso anomalo ma regolare dei servizi.

Tracciabilità La tracciabilità riguarda la registrazione degli eventi per poter risalire ai relativi responsabili. I file in cui vengono salvate le tracce sono detti *secure audit trail/log*. Anche questi file possono essere manipolati se il sistema viene compromesso, perciò vanno trattati adeguatamente (server separato, file di sola lettura, ...).
Una proprietà più stringente della tracciabilità è detta **non-repudiation**, in cui l'autore di un'azione non può rinnegare di averla compiuta.

Autenticazione Dati e servizi devono risultare disponibili solo per utenti autorizzati, in base alle policy definite nel sistema. È necessario dunque verificare l'identità di utenti o sistemi, sfruttando uno o più tra i seguenti metodi:

- Qualcosa che abbiamo (es: documento)
- Qualcosa che conosciamo (es: password)
- Qualcosa che siamo (es: impronte digitali)

1.2 Sicurezza come risk minimization

La sicurezza informatica si può definire come la protezione delle risorse (dette *asset*) dalle minacce, o *threats*, possiamo dunque decidere di scegliere

politiche e meccanismi di sicurezza in base al valore delle risorse ed al rischio di attacco. Questo approccio è ben definito nella pratica, e permette di stabilire con certificati standard la sicurezza del sistema. Valutando le varie minacce, in base alla probabilità di accadimento e all'eventuale impatto sul sistema, possiamo progettare contromisure ad hoc per ciascuna.

$$\texttt{Rischio} = \texttt{Possibilità di accadere} \times \texttt{Impatto sul sistema}$$

Non possiamo proteggerci da ogni possibile minaccia, in base al fattore di rischio decidiamo se prendere contromisure ed eventualmente quanto investire in quest'ultime.

Quando parliamo in minacce trattiamo sia eventi accidentali sia comportamenti malevoli effettuati da parte di *attaccanti*. Studiamoli dettagliatamente.

Attaccanti Gli attaccanti, o *threat agents*, sono utenti che adottano un comportamento malevolo e si possono distinguere in base alle seguenti categorie:

- Dipendenti che effettuano azioni dannose involontariamente.
- Hacker guidati da sfida personale, usualmente privi di volontà di nuocere.
- Ex dipendenti, magari un po' astiosi.
- Criminali, terroristi, agenti di spionaggio esteri.

Dalla definizione di attaccante ne derivano altre due di pari importanza: *vulnerabilità* e *superficie di attacco*.

Vulnerabilità Debolezza del sistema che può essere sfruttata da una minaccia per arrecare un danno. Le vulnerabilità possono essere dovute all'ambiente fisico (hardware accessibile agli attaccanti), software o di rete. Nel caso una vulnerabilità non sia tuttora nota allo sviluppatore viene detta *zero-day*.

Superficie di attacco Parti del sistema esposte a manipolazione da parte di utenti non autorizzati. Tale superficie dev'essere minimizzata per incrementare la sicurezza del sistema. Ad esempio, in ambiente di rete è rischioso lasciare aperte porte non utilizzate dal sistema.

Fasi per l'analisi e la riduzione dei rischi:

1. Analisi dei rischi esistenti:
   - Identificazione degli asset da proteggere.
   - Identificazione dei rischi in base a minacce e vulnerabilità.
2. Analisi delle soluzioni di sicurezza proposte e dei trade-off che ne derivano.

1.3 Ingegneria della sicurezza

Far si che il sistema si comporti nei modi specificati → *Ingegneria del SW.*

Impedire che il sistema si comporti in modi non specificati → *Ingegneria della sicurezza.*

Per rendere il sistema sicuro è necessario un approccio *olistico*: la sicurezza deve essere progettata insieme al sistema, non aggiunta successivamente!

L'implementazione di una security solution prevede: analisi e modellazione dei rischi, specifica di security policy, progettazione di adeguate contromisure ed infine studio dei trade-off per decidere quali meccanismi adoperare al fine di rendere sicuro il sistema.
Risultano fondamentali i seguenti punti:

- Chiarire e documentare gli obiettivi di sicurezza prima dello sviluppo.
- Ridurre la complessità mediante tecniche di sviluppo adeguate.
- Verificare accuratamente che ogni requisito di sicurezza sia soddisfatto.

1.4 Metodi formali

Tecniche e strumenti basati su matematica e logica impiegati nelle fasi di specifica, costruzione ed analisi di sistemi hardware e software. Attraverso i metodi formali è possibile scrivere documentazione non ambigua affinché sia più efficente evitare o almeno individuare errori nell'implementazione. Un modello di sicurezza formale ha il vantaggio di poter distinguere *quali requisiti* il sistema deve poter soddisfare da *come* esattamente esso li soddisfi.

2. Meccanismi ed algoritmi crittografici

Vogliamo trasferire delle informazioni tra due entità: come trasformiamo canali inaffidabili in *canali affidabili*? Prima di tutto dobbiamo definire quali proprietà devono essere soddisfatte dal canale, ad esempio:

- Confidenzialità: i dati trasmessi rimangono segreti.
- Integrità: le informazioni non sono state alterate.
- Autenticazione: i principal conoscono le reciproche identdità.

La *tecnologia abilitante* per ottenere tali risultati è la *crittografia*.

Figura 2.1: Schema crittografico generico.

Termini fondamentali:

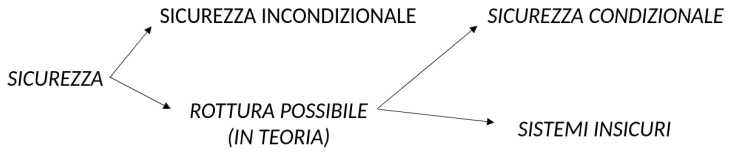
- *Crittologia*: Studio della scrittura segreta.
- *Steganografia*: Scienza del nascondere messaggi in altri messaggi.
- *Crittografia*: Scienza della scrittura segreta.
- *Crittoanalisi*: Scienza del recupero del testo originale da un testo cifrato senza conoscere la chiave di decifratura.

In uno schema generico, come in figura [2.1], abbiamo due agenti A e B che si scambiano un messaggio in chiaro M (anche detto *plaintext* P), cifrandolo con una chiave K_1 e decifrandolo con una chiave K_2, ed indichiamo il testo cifrato con C. Distinguiamo principalmente due casi:

- Crittografia a chiave simmetrica: $K_1 = K_2$, oppure l'una è facilmente ricavabile conoscendo l'altra.
- Crittografia a chiave asimmetrica/pubblica: $K_1 \neq K_2$ e non è possibile usare una delle due chiavi per ricavare l'altra.

Se le chiavi sono note, le fasi di cifratura e decifratura dovrebbero essere computazionalmente facili da effettuare.
NB: *La sicurezza dipende dalla segretezza della chiave, non dell'algoritmo.*

Unconditional security Il sistema è sicuro anche se l'avversario disponesse di potenza computazionale illimitata. Si basa sulla *teoria dell'informazione*.



Conditional security Il sistema è teoricamente a rischio, tuttavia per romperlo l'avversario avrebbe bisogno di una potenza computazionale irrealisticamente elevata.

2.1 Crittoanalisi

Obiettivo del crittoanalista è ottenere il testo originale, tuttavia usualmente l'attaccante è interessato anche a recuperare la chiave. Vediamo due approcci comuni: attacco brute-force e attacco di crittoanalisi.

Brute-force attack L'attacco forza-bruta è sempre attuabile e consiste semplicemente nel provare a decifrare il ciphertext con ogni chiave possibile, ammesso di conoscere l'algoritmo e di poter riconoscere il testo originale una volta ottenuto. La complessità cresce esponenzialmente rispetto alla dimensione della chiave.

Attacchi di crittoanalisi Da ora in poi assumeremo sempre che l'algoritmo sia noto all'attaccante. Possiamo distinguere vari tipi di attacco in base alle informazioni o ai servizi a cui può accedere l'attaccante:

- *Solo testo cifrato*: dati $C_1 = E_K(M_1), \ldots, C_n = E_K(M_n)$, l'attaccante può dedurre $M_1, \ldots, M_n$ oppure un algoritmo per recuperare M_{n+1} dato $E_K(M_{n+1})$.
- *Known plaintext*: dati $M_1, \ldots, M_n$ e relativi $C_1, \ldots, C_n$, l'attaccante può dedurre la chiave di decifratura oppure un algoritmo per recuperare M_{n+1} dato $E_K(M_{n+1})$.
- *Chosen plaintext*: uguale al known plaintext, ma in questo caso l'attaccante può scegliere arbitrariamente i testi in chiaro $M_1, \ldots, M_n$.
- *Adaptive chosen plaintext*: uguale al chosen plaintext, ma in questo caso l'attaccante può modificare i testi in chiaro da sottoporre a cifratura in base ai risultati delle cifrature precedenti.
- *Chosen cyphertext*: L'attaccante può scegliere diversi testi cifrati da decifrare e vederne quindi il testo in chiaro.

Definizione di sicurezza

Vogliamo garantire una proprietà di sicurezza per il sistema. Per certificare formalmente la soddisfazione dei requisiti di tale proprietà, costruiamo una

definizione di sicurezza sfruttando il modello in figura [2.2]. Il procedimento prevede le seguenti fasi:

1. Specificare un oracolo (ovvero il tipo di attacco).
2. Definire la condizione di vittoria del gioco per l'attaccante, ovvero una condizione sull'output.
3. Il sistema è *sicuro* rispetto a quel determinato tipo di attacco se nessun avversario efficente può vincere il gioco con probabilità non trascurabile.

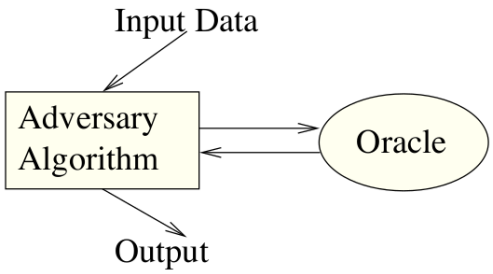


Figura 2.2: Modello per costruire una definizione di sicurezza.

Esempio di definizione di sicurezza: *conventional encryption*.

- Nessun dato in input.
- L'avversaro può fare chosen plaintext su un messaggio scelto M o su un testo scelto casualmente.

IDEA: Se l'avversario non riesce a distinguere un testo casuale cifrato rispetto al testo cifrato scelto arbitrariamente, non può arrecare alcun danno in casi reali.

Formalizzazione matematica

`Notazione`:

- $\mathcal{A}$: l'alfabeto, insieme finito.
- $\mathcal{M} \subseteq \mathcal{A}^*$: spazio dei messaggi $\longrightarrow M \in \mathcal{M}$: plaintext.
- $\mathcal{C}$: spazio dei messaggi cifrati, il cui alfabeto può essere diverso da $\mathcal{A}$.
- $\mathcal{K}$: spazio delle chiavi.
- Data $e \in \mathcal{K}$, definiamo E_e: funzione di cifratura con chiave e (*encryption function*).
- Data $d \in \mathcal{K}$, definamo D_d: funzione di decifratura con chiave d (*decryption function*).

Schema di cifratura (o cifrario) Consiste di un insieme $\{E_e \mid e \in K\}$ ed un corrispondente insieme $\{D_d \mid d \in K\}$ aventi la seguente proprietà:

$$\forall e \in K \,.\, \exists!\, d \in K \quad \text{tali per cui} \quad D_d = E_e^{-1}$$

ovvero:

$$D_d(E_e(M)) = M, \quad \forall M \in \mathcal{M}$$

Le chiavi e e d formano una *coppia di chiavi* (e, d).

2.2 Cifratura a chiave simmetrica

Uno schema di cifratura è detto *a chiave simmetrica* se per ogni coppia di chiavi (e, d) è computazionalmente facile ricavare d da e o viceversa. In pratica: $e = d$. Due principal conoscono una chiave condivisa con cui cifrare/decifrare i messaggi scambiati. Costituiscono tuttora i protocolli crittografici più usati, in quanto risultano molto veloci.
Per cifrare messaggi di lunghezza arbitraria vengono usati diversi metodi:

- **Block cipher**: il plaintext viene diviso in blocchi di lunghezza fissa ed ogni blocco viene cifrato separatamente.
- **Stream cipher**: block cipher con lunghezza pari ad 1.
- **Codes**: ad ogni parola è associato un codice identificativo.

Studiamo ora diversi tipi di cifratura che sfruttano tecniche di sostituzione, trasposizione o combinazioni di esse.

Substitution ciphers

L'idea alla base di questi cifrari consiste nel sostituire ogni lettera del messaggio secondo determinate regole.

- *Cifrario di Cesare*: ogni lettera viene "trascinata" di un certo numero di posizioni. La chiave è proprio il numero di posizioni.
  `Debolezza`: brute-force.
- *Mono-alfabetico*: generalizzazione del cifrario di Cesare. La chiave è una delle 26! permutazioni dell'alfabeto e si associa ad ogni lettera un'altra lettera arbitraria.
  `Debolezza`: analisi delle frequenze.
- *Sostituzione omofonica*: sostituisco a con una delle stringhe dell'insieme $H(a)$, in questo modo l'analisi delle frequenze è più difficile ma si complica anche la fase di decifratura.
- *Vigenère*: usa la stessa idea del cifrario di Cesare, ma la chiave è composta da più chiavi, es: $K = K_1K_2K_3$. La chiave viene ripetuta sul testo ed ogni lettere del plaintext è cifrata in base alla chiave corrispondente a tale posizione.
  `Debolezza`: analisi delle frequenze con lunghezza della chiave crescente.
- *Poli-alfabetico*: usa la stessa idea del cifrario di Vigenère ma ad ogni blocco associa una permutazione diversa invece di uno shift. La chiave è costituita dalla dimensione dei blocchi e dalle permutazioni.

- *One-time pad*: il cifrario di Vernam è uno stream cipher con $\mathcal{A} = \{0, 1\}$. Effettua uno xor tra plaintext e chiave (di lunghezza pari a quella del messaggio). Il ciphertext non contiene alcuna informazione sul messaggio, ma è difficile per i principals scambiarsi una chiave di tale lunghezza.

Trasposition ciphers

Le lettere del messaggio originale non vengono sostituite, bensì solo scambiate di posizione. I cifrari basati su trasposizione sono spesso rompibili attraverso analisi delle frequenze.

Classico esempio è la scitala greca: il testo veniva scritto su una cintura ed era comprensibile solo se tale cintura veniva avvolta intorno ad un bastone dell'esatto diametro.

Cifrari compositi

Due sostituzioni poste in cascata sono in realtà una sola sostituzione, così come due trasposizioni in cascata.
Sostituzione seguita da trasposizione → nuovo cifrario più complesso. Nel 1949 Shannon introduce le substitution-permutation network:

- Substitution → confusione del messaggio (nasconde i collegamenti tra plaintext e ciphertext)
- Permutation → diffonde il messaggio (impedisce l'analisi statistica)

Cifrario di Feistel: partiziona il plaintext in due metà (HL, HR), esegue n round di cifratura ed infine le concatena nuovamente per formare l'effettivo ciphertext. Ad ogni round:

1. applica ad HR una round-function con una chiave;
2. effettua lo xor con HL;
3. scambia HL con HR.

DES: Data Encryption Standard. Variante di Feistel a 16 round, lavora con chiavi a 56 bit dividendo il testo in blocchi da 64 bit. Possibili atticchi: brute-force o attacco matematico basato su critoanalisi lineare (riduce lo spazio delle chiavi da 2^{56} a 2^{43}).

Vorremmo aumentare la sicurezza di DES per evitare che venga rotto mediante brute-forcing. Se mettiamo due DES in cascata? Double-DES: Non funziona, debole all'attacco *meet-in-the-middle* che riduce lo spazio delle chiavi a 2^{56}.
Triple-DES: tre DES con due chiavi, K_1 e K_2, posti in cascata in modo non banale, ovvero:

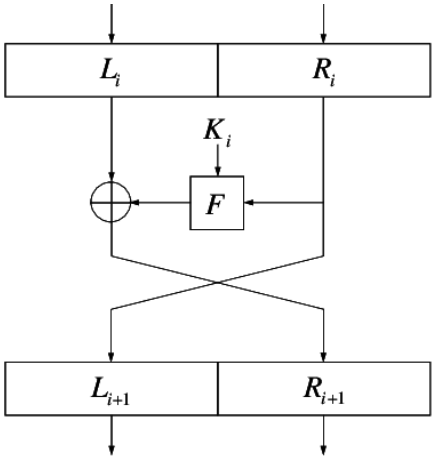


Figura 2.3: Round di cifratura di Feistel.

- Encryption: primo e terzo DES cifrano usando K_1, mentre il secondo DES decifra con chiave K_2.
- Decryption: primo e terzo DES decifrano usando K_1, mentre il secondo DES cifra con chiave K_2.

Se $K_1 = K_2$, triplo DES è compatibile con DES. Unico attacco possibile: brute-force con spazio delle chiavi 2^{112}.

AES: Advanced Encryption Standard. Evoluzione di triple-DES, in quanto troppo lento e basato su blocchi a 64 bit. AES non ha una struttura di Feistel e lavora con blocchi da 128, 192 o 256 bit.

Plaintext di dimensioni maggiori ad un blocco Come può essere applicato un cifrario a blocchi su un messaggio la cui lunghezza eccede le dimensioni del blocco? Esistono molte soluzioni, vediamone solo due:

- *Elettronic codebook mode.*
  Messaggio spaccato in m blocchi che sono cifrati separatamente.
  `Debolezze`: a blocchi in chiaro uguali corrispondono blocchi cifrati uguali; difficile verificare alterazioni del messaggio cifrato.
- *Cipher-block chaining.*
  Messaggio spaccato in m blocchi. Prima di cifrare un blocco, viene effettuato lo xor di tale blocco con il testo cifrato di quello precedente:

  Encryption: $C_i = E_k(P_i \oplus C_{i-1})$
  Decryption: $P_i = C_{i-1} \oplus D_k(C_i)$

  La struttura a catena permette il verificarsi di due importanti proprietà:

  1. Blocchi uguali sono cifrati in modo diverso.

2. Se C_j viene alterato ma C_{j+1} no, i blocchi successivi vengono decifrati correttamente.

2.3 Posizionamento dell'encryption

Link Encryption	End-to-End Encryption
La cifratura è applicata ad ogni link.	La cifratura è applicata solo tra sorgente e destinazione.
Trasparente all'utente.	L'utente deve implementare il protocollo crittografico.
Tutto viene cifrato.	L'utente sceglie cosa cifrare.
Cifratura applicabile anche a livello HW.	Cifratura applicabile solo a livello SW.
Richiede diverse coppie di chiavi.	Chiavi condivise solo tra le due parti della comunicazione.
Protegge dal monitoring del traffico.	*Protegge i dati mantenendoli cifrati.*

2.4 Crittografia a chiave asimmetrica

La crittografia a chiave asimmetrica, o a chiave pubblica, viene introdotta come soluzione a due problemi: *distribuzione delle chiavi* e *firma digitale*. Per spiegarne il funzionamento usiamo il classico esempio dell'invio di un messaggio da A a B.

1. A vuole inviare un messaggio M a B.
2. B crea una coppia di chiavi: (PU_b, PR_b), PU_b è detta *chiave pubblica*, PR_b è detta *chiave privata*.
3. A cifra M con la chiave pubblica di B, inviando C

$$C = E(PU_b,\, M)$$

4. B decifra C usando la propria chiave privata, recuperando M

$$M = D(PR_b,\, C) = D(PR_b,\, E(PU_b, M))$$

Affinché la crittografia a chiave pubblica sia effettivamente sicura, si rendono necessari alcuni *requisiti*: i passi 2, 3 e 4 devono essere computazionalmente facili da eseguire, mentre dev'essere impraticabile per un avversario:

- Determinare PR_b conoscendo PU_b.
- Determinare M conoscendo PU_b ed esempi di testi cifrati.

Simmetric encryption	Public-key encryption
`Requisiti di funzionamento`	
Unici algoritmo e chiave per cifrare e decifrare.	Algoritmi e chiavi diversi per cifrare e decifrare.
I due utenti condividono la chiave e l'algoritmo.	L'utente che invia il messaggio conosce solo la chiave pubblica e l'algoritmo.
`Requisiti di sicurezza`	
La chiave deve rimanere segreta.	La chiave privata deve rimanere segreta.
Dev'essere impossibile o impraticabile decifrare un messaggio senza ulteriori informazioni.	Dev'essere impossibile o impraticabile decifrare un messaggio senza ulteriori informazioni.
La conoscenza dell'algoritmo e di testi cifrati dev'essere insufficente a determinare la chiave.	La conoscenza dell'algoritmo, della chiave pubblica e di testi cifrati dev'essere insufficente a determinare la chiave privata.

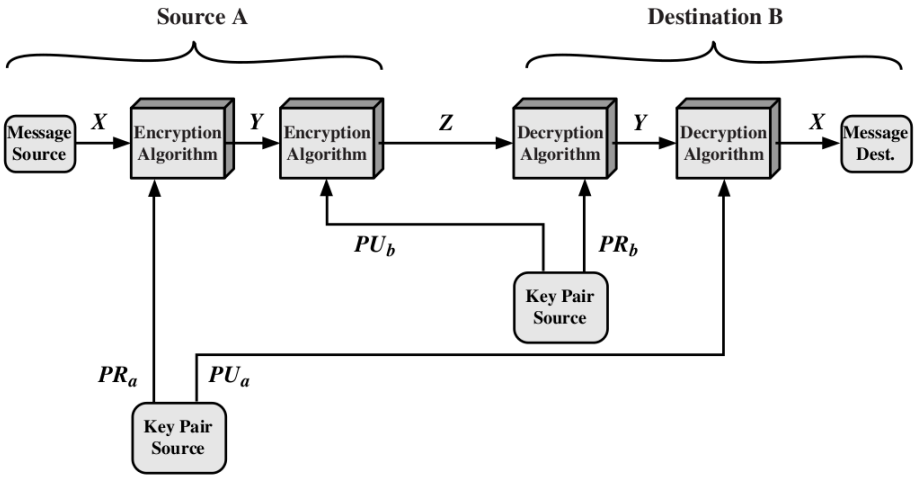


Figura 2.4: Crittografia a chiave pubblica usata per garantire confidenzialità e autenticazione.

Algorithm	Encryption/ Decryption	Digital Signature	Key Exchange
RSA	Yes	Yes	Yes
Elliptic Curve	Yes	Yes	Yes
Diffie-Hellman	No	No	Yes
DSS	No	Yes	No

Figura 2.5: Soddisfazione dei requisiti di sicurezza dei principali protocolli.

Funzioni one-way trapdoor

Una funzione $f : X \to Y$ è detta *one-way* se f è "facile" da computare $\forall x \in X$, mentre f^{-1} è "difficile".
Una funzione $f_k : X \to Y$ è detta *one-way trapdoor* se, data l'informazione extra k (*trapdoor information*), è praticabile trovare una $x \in X$ per $y \in Im(f)$ tale che $f_k(x) = y$.

RSA - Rivest, Shamir, Adleman

RSA è l'algoritmo crittografico a chiave pubblica più popolare e basa la sua sicurezza sulla difficoltà nel fattorizzare numeri di grandi dimensioni, le chiavi sono infatti numeri primi di oltre 100 cifre. Problema: computazionalmente pesante, per essere sicuro richiede chiavi da 1024 o 2048 bit.

Funzionamento Di seguito sono descritti i passi dell'algoritmo, per comprendere i quali è necessario conoscere alcune basi di teoria dei numeri (vedi appendice).

- Generazione delle chiavi:
  1. Generiamo 2 numeri primi distinti di almeno 100 cifre: p e q.
  2. Computiamo $n = pq$ e $\phi = (p-1)(q-1)$.
  3. Scegliamo e tale che $1 < e < \phi$, con e, ϕ coprimi.
  4. Computiamo $d = e^{-1} \, mod \, \phi$.
  5. Pubblichiamo $PU = (e, n)$ e manteniamo segreta $PR = (d, n)$.
- Cifratura con chiave (e, n):
  1. Dividiamo M in $\log_2(n)$ blocchi: $M_1 M_2 \dots M_l$, con $M_i < n$.
  2. Computiamo $C_i = M_i^e \, mod \, n$.
- Decifratura con chiave (d, n):
  1. Computiamo $M_i = C_i^d \, mod \, n$.

L'algoritmo funziona in quanto, se la coppia (e, d) viene scelta in modo corretto, valgono le seguenti equazioni:

$$C = M^e \, mod \, n$$
$$M = C^d \, mod \, n = (M^e)^d \, mod \, n = M^{ed} \, mod \, n$$

Sicurezza di RSA La sicurezza dell'algoritmo dipende dalla complessità nella risoluzione del problema della fattorizzazione. Non si conoscono algoritmi per risolverlo in tempo polinomiale, tuttavia per rendere impraticabile il calcolo di d dati e ed n è necessario usare chiavi da almeno 1024 bit.

Problema della distribuzione delle chiavi

Gli algoritmi a chiave simmetrica sono nettamente più efficenti di quelli a chiave pubblica, tuttavia per funzionare richiedono che mittente e destinatario conoscano la chiave crittografica. Soluzione: usare un algoritmo a chiave pubblica per scambiarsi una *chiave di sessione*, con la quale cifrare i messaggi scambiati mediante protocolli crittografici simmetrici.
Studiamo i seguenti approcci:

- RSA
- Diffie-Hellman
- El Gamal
- Massey-Omura

Scambio di chiavi con RSA

- Per cifrare M con $PU = (e, n)$, scegliamo k casualmente:

  $$C = (k^e \, mod \, n, \, E_k(M))$$

- Per decifrare C con $PR = (d, n)$, lo dividiamo in (C_1, C_2):

  $$k = C_1^d \, mod \, n \qquad M = D_k(C_2)$$

I messaggi possono in seguito essere scambiati con crittografia simmetrica con chiave k per una maggior efficenza.
Problema: se d viene compromessa in futuro ed un attaccante ha registrato la conversazione, può recuperare i messaggi cifrati.

Scambio di chiavi con Diffie-Hellman Per poter comprendere il protocollo DH è necessario avere un background riguardo i *logaritmi discreti* (vedi appendice).
`Funzionamento`:

1. I principals si scambiano un numero primo q e la *radice primitiva* α.
2. A genera X_A e B genera X_B, entrambe inferiori ad q.
3. A genera $Y_A = \alpha^{X_A} \, mod \, q$ e B genera $Y_B = \alpha^{X_B} \, mod \, q$.

4. A invia Y_A a B mentre B invia Y_B ad A.
5. A computa $K_A = Y_B^{X_A}$ e B computa $K_B = Y_A^{X_B}$.
   Le chiavi K_A e K_B sono uguali:

$$\begin{aligned} K_A &= Y_B^{X_A} \, mod \, q \\ &= (\alpha^{X_B})^{X_A} \, mod \, q \\ &= (\alpha^{X_A})^{X_B} \, mod \, q \\ &= Y_A^{X_B} \, mod \, q = K_B \end{aligned}$$

Proprietà di Diffie-Hellman:

+ Nessuno dei due agenti può sabotare la chiave condivisa generata.
+ *Perfect Forward Secrecy* (PFS): la chiave di sessione non è compromessa anche se una chiave privata viene scoperta in futuro.
− Le chiavi non sono autenticate: possibile man-in-the-middle.

Man-in-the-middle attack:

1. A trasmette Y_A a B.
2. Z intercetta Y_A e trasmette Y_Z a B.
   Inoltre, Z calcola $K_1 = (Y_A)_Z^X \, mod \, q$.
3. B riceve Y_Z e calcola $K_2 = (Y_Z)_B^X \, mod \, q$.
4. B trasmette Y_B ad A.
5. Z intercetta Y_B e trasmette Y_Z ad A.
   Inoltre, Z calcola $K_2 = (Y_B)_Z^X \, mod \, q$.
6. A riceve Y_Z e calcola $K_2 = (Y_Z)_A^X \, mod \, q$.

A questo punto, A e B credono di condividere una chiave, in realtà A condivide K_1 con Z e B condivide K_2 con Z.

El Gamal Variante di Diffie-Hellman. Partendo dal medesimo setup (q numero primo e α radice primitiva), sfrutta inoltre una funzione crittografica simmetrica E:

1. B sceglie X_B, calcola $Y_B = \alpha^{X_B} \, mod \, q$ e lo invia ad A.
2. A sceglie X_A, calcola $Y_A = \alpha^{X_A} \, mod \, q$, calcola la chiave $K = Y_B^{X_A} \, mod \, q$ ed invia a B questo messaggio: $(E(M, K), Y_A)$.
3. B calcola $K = Y_A^{X_B} \, mod \, q$ ed usa K per decifrare $E(M, K)$.

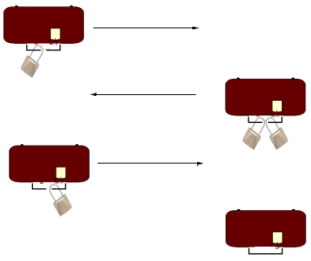
Massey-Omura Schema di cifratura privo di chiavi condivise, basato sul problema del logaritmo discreto. Il principio per cui funziona è il seguente:

1. I principal si scambiano un numero primo p e ognuno dei due sceglie due valori, e e d, tali che $ed \, mod \, (p-1) = 1$.
2. Esiste quindi k tale per cui $ed = k(p-1) + 1$.
3. Per il teorema di Eulero, per ogni $m \in \{1, \ldots, p-1\}$ vale:

$$M^{ed} \, mod \, p = M^{k(p-1)} M \, mod \, p = M \, mod \, p = M$$

Funzionamento del protocollo:

1. A $\rightarrow$ B: $M^{e_A} \, mod \, p$
2. B $\rightarrow$ A: $M^{e_A e_B} \, mod \, p$
3. A $\rightarrow$ B: $M^{e_A e_B d_A} \, mod \, p \; (= M^{e_B})$
4. B $\rightarrow$ A: $M^{e_A e_B d_A d_B} \, mod \, p \; (= M)$



Integrità del messaggio

I dati non possono essere alterati o manipolati da entità non autorizzate, in caso contrario abbiamo gli strumenti per rilevare l'eventuale modifica. Nei sistemi operativi questa proprietà è garantita dal *controllo degli accessi*, in ambito di rete dobbiamo utilizzare la crittografia.
IDEA: sfruttare le *funzioni di hash* per creare un'impronta del messaggio. Una funzione di hash $h(x)$ possiede queste proprietà:

- *Compressione*: l'output di $h(x)$ è una stringa di lunghezza fissa.
- *Efficenza*: $h(x)$ può essere computata in tempo polinomiale.

$h(x)$ è una funzione di hash *crittografica* se è inoltre:

- *One-way function* (o pre-image resistant).
- *2-nd pre-image resistant*: dato x, è difficile trovare x' *t.c.* $h(x) = h(x')$.
- *Collision resistant*: è difficile trovare x, x' *t.c.* $h(x) = h(x')$.

Il valore di hash è detto *message digest* o *modification detection code*.

Un classico impiego delle funzioni hash si ha nei file per lo storing di password, per i quali è sufficiente che la funzione h soddisfi la proprietà di pre-image resistance in quanto combinata con il *sale*.

Autenticazione del messaggio

La *message authentication* (detta anche data-origin authentication) comprende l'autenticazione del mittente del messaggio e l'integrità dello stesso. Le principali tecniche per l'autenticazione del messaggio sono *Message Authentication Code* (MAC) e *firma digitale*.

MAC Un algoritmo per il Message Authentication Code consiste in una famiglia di funzioni hash h_k parametrizzate da una chiave segreta k. Queste funzioni h_k devono essere *computation resistant*: date zero o più coppie di MAC $(x_i, h_k(x_i))$, dev'essere impraticabile computare una nuova coppia $(x, h_k(x))$ con $x \neq x_i$ senza conoscere la chiave k.

Attravarso il MAC possiamo garantire message authentication, ma non la non-repudiation né la freshness. La prima non è verificata in quanto il destinatario B potrebbe creare il messaggio ed applicarvi il MAC fingendo sia stato A ad inviarlo. La *freshness* è una proprietà che indica che il messaggio

ricevuto è nuovo, ovvero non è la copia di un vecchio messaggio intercettato da un attaccante e rispedito in seguito (*replay attack*). Il protocollo MAC non garantisce quest'ultima proprietà.

Firma digitale La firma digitale (*digital signature*) è un protocollo fondamentale per garantire autenticazione e non-repudiation. Studiamone il funzionamento. `Notazione`:

- $\mathcal{M}$: spazio dei messaggi.
- $\mathcal{S}$: spazio delle firme, ovvero stringhe di lunghezza fissa (n-bit).
- $S_A \colon \mathcal{M} \to \mathcal{S}$: creazione della firma per A, tenuta segreta.
- $V_A \colon \mathcal{M} \times \mathcal{S} \to \{true, false\}$: verifica pubblica della firma di A.

Dati S_A e V_A si può costruire uno schema di firma per A:

- *Procedura di firma*
  A crea una firma per $M \in \mathcal{M}$, $S = S_A(M)$, e trasmette (M, S).
- *Procedura di verifica*
  B verifica la firma di A, $U = V_A(M, S)$, ed accetta il messaggio se e solo se $U = true$.
- *Requisiti di sicurezza*
  Per chiunque eccetto A dev'essere difficile, dato $M \in \mathcal{M}$, trovare $S \in \mathcal{S}$ tale che $V_A(M, S) = true$.

Il meccanismo di firma digitale può essere basato su sistemi crittografici a chiave pubblica *reversibili*, ovvero i sistemi per cui vale $\mathcal{M} = \mathcal{C}$ e vale quindi:

$$D_d(E_e(M)) = E_e(D_d(M)) = M \qquad \forall M \in \mathcal{M}$$

Costruiamo lo schema per la firma digitale:

1. Siano $\mathcal{M}$ e $\mathcal{C}$ gli spazi di messaggi e firme, $\mathcal{M} = \mathcal{C}$.
2. Sia (e, d) una coppia di chiavi per lo schema di cifratura a chiave pubblica.
3. Definiamo la funzione di firma S_A per essere D_d, ovvero $S = D_d(M)$.
4. Definiamo V_A come:

$$V_A(M, S) = \begin{cases} true, & \text{se } E_e(S) = M \\ false, & \text{altrimenti} \end{cases}$$

Problema: questo schema è vulnerabile ad un **forgery attack**:

1. L'attaccante Z sceglie casualmente una firma S e computa $M = E_e(S)$.
2. Dato che $\mathcal{C} = \mathcal{M}$, B può inviare (M, S) come messaggio firmato.
3. La verifica ritorna $true$ anche se A non ha firmato il messaggio.

Soluzione:

- Nel messaggio viene nominato il mittente.

- Firma dell'hash del messaggio al posto dell'intero messaggio.

$$\begin{aligned} \text{Firma: } & (M,\, D_d(h(M))) \\ \text{Verifica: } & h(M) = E_e(D_d(M)) \end{aligned}$$

La coppia messaggio-firma può essere cifrata per garantire confidenzialità.

3. Key management

Quando trattiamo di *key management* ci riferiamo alle infrastrutture che permettono di gestire:

- la distribuzione di chiavi crittografiche;
- i meccanismi di associazione chiave-identità;
- la generazione, il mantenimento e la revoca delle chiavi.

Le *Public-Key Infrastructures* (PKI) permettono ai principals di riconoscere i proprietari delle chiavi. Per entrare a far parte della PKI, l'utente A deve:

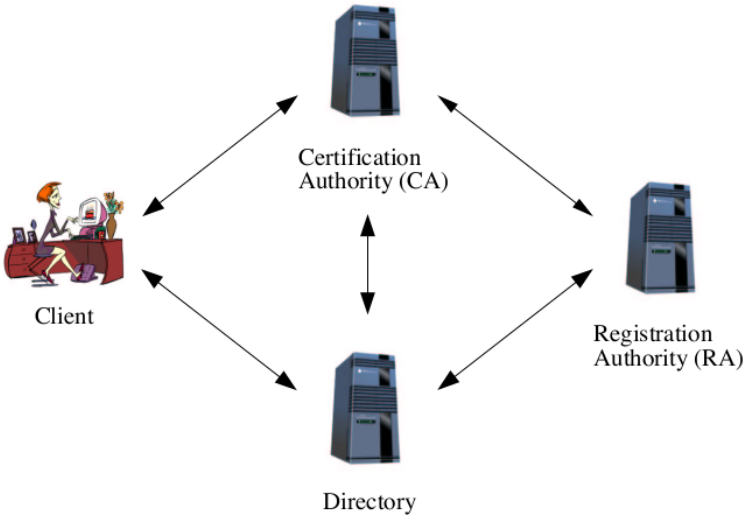
1. Generare una coppia di chiavi, (PR_A, PU_A).
2. Consegnare PU_A ad una *Certification Authority* (CA).

La CA *verifica* l'identità di A e *firma* un certificato che asserisce:

> "La chiave PU_A appartiene ad A."

La mutua fiducia nella certification authority permette a due principals di avere un canale sicuro.

3.1 Componenti di una PKI



Una PKI può essere *closed* se è limitata ad un gruppo specifico di utenti (es: VPN), oppure *open*. Indipendentemente dal tipo, le componenti di una public-key infrastructure sono: clients, CA, *Registration Authority* (RA) ed una directory.

- *Certification Authority*:
  - Crea certificati e li pubblica nella directory.
  - Mantiene la *Certificate Revocation List* (CRL) nella directory.
  - Esegue il backup di alcune chiavi per recuperi futuri.
- *Directory*:
  - Server contenente certificati e CRL.
  - Identifica univocamente gli utenti.
  - Dev'essere altamente disponibile.
- *Registration Authority*:
  - Gestisce la registrazione e l'identificazione degli utenti.
  - Rilascia i nuovi certificati.
- *Clients*:
  - Autenticazione (one-way, two-way).
  - Confidenzialità dei messaggi con schema a chiave pubblica.
  - Firma digitale.

3.1.1 Certificate revocation and recovery

Le CRL sono firmate e mantenute dalla CA nella directory, affinché gli utenti possano controllare che un determinato certificato non sia stato revocato. La revoca può scaturire in caso di possibili compromissioni di chiavi private o pubbliche o se l'utente non è più certificato dalla CA.
Una CRL contiene:

- Nome dell'emittente
- Data di creazione della CRL
- Data di rilascio della prossima CRL
- Una entry per ogni certificato revocato

Le CA permettono anche di recuperare le chiavi private, salvandole sui propri server. Ciò può rivelarsi pericoloso, ma è spesso importante averne memorizzata una copia. Mediante *key escrow* una terza parte può recuperare la chiave, spesso per questioni legali.

3.1.2 Naming and identity

Un principal è un'entità univoca, mentre una *identity* è una sua rappresentazione. Compito di una PKI è creare un legame tra un principal e la sua relativa identity. Quest'ultima è difficile da definire, in quanto vogliamo delle policies per l'*assegnamento* e la *risoluzione* dei nomi.
X.509 sfrutta i *distinguished names* (DNs) per identificare sia gli utenti che gli emittenti e richiedono la soddisfazione di due criteri:

- *Permanenza*: i DNs non devono contenere informazioni volatili.
- *Unicità*: i DNs devono essere unici nel tempo.

Le CA garantiscono, a certi livelli, per l'identità dei principals a cui viene emesso il certificato:

- *CA Authentication policy*
  Descrive il livello di autenticazione richiesto per identificare il principal a cui viene emesso il certificato. Stabilisce il livello di dimostrazione di identità richiesto dalla CA per accettare l'affermazione di identità del principal.
- *CA issuance policy*
  Descrive i principals a cui la CA permette di certificarsi.
  (Data l'identità del soggetto, è idoneo alla certificazione?)

3.2 Certificati

Un certificato è un *token* che collega un'identità ad una chiave e deve essere firmato da una CA. Il certificato C_A contiene la chiave pubblica dell'utente A, una rappresentazione della sua identità ed un timestamp T:

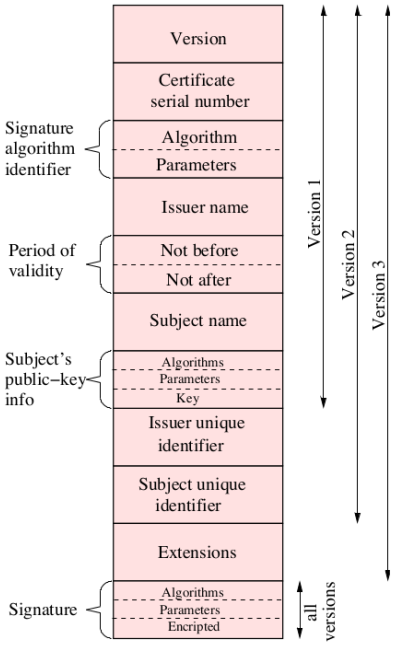
$$C_A = M||E(PR_{CA},\, H(M)) \qquad \text{dove } M = \langle PU_A,\, A,\, T\rangle$$

Come modello di certificato studiamo lo standard *X.509*, su cui si basano struttura e protocolli di autenticazione usati in vari contesti (es: IPSEC, SSL/TLS, ...). Tale standard si basa su crittografia a chiave pubblica (raccomandato RSA), funzioni hash e firma digitale.

3.2.1 Modello X.509

Componenti del certificato:

- *Versione del certificato*
- *Numero seriale*
  La coppia *emittente - numero seriale* identifica univocamente il certificato.
- *Signature algorithm identifier*
  Algoritmo usato per firmare il certificato.
- *Issuer name*
  Identificativo della CA.
- *Periodo di validità*
- *Nome del soggetto*
- *Informazioni sulla chiave pubblica*
  Algoritmo, parametri e chiave del soggetto.
- *Firma*
  Contiene l'hash degli altri campi, firmato dalla CA.



Certificato di A firmato dalla CA:

$$CA\langle\langle A\rangle\rangle = M \,||\, E(PR_{CA},\, H(M)), \quad \text{con } M = \langle V,\, SN,\, AI,\, CA,\, T_A,\, A,\, Ap\rangle$$

Chi intende comunicare con A richiede questo certificato alla CA, calcola l'hash dei campi di M e lo confronta con quello contenuto nel certificato, ricavato usando la chiave pubblica della CA.

3.3 Modelli di fiducia

I modelli di fiducia permettono di gestire il modo in cui gli utenti stabiliscono la validità di un certificato.

- *Modello diretto*
  Tutti gli utenti sono iscritti alla stessa CA e vi ripongono fiducia.
- *Modello gerarchico*
  Distribuzione ad albero delle CA, ad ognuna delle quali corrisponde una frazione di utenti ai quali consegna la propria PU in modo sicuro. Gli utenti ripongono fiducia nei certificati di *root*, mentre per validare certificati di CA non root occorre usare una *chain of certificates*:

  $$X_1\langle\langle X_2\rangle\rangle X_2\langle\langle B\rangle\rangle$$

Per comunicare con utenti relativi a CA diverse si applica la *cross-certification*, ovvero le CA si garantiscono reciprocamente.
X.509 suggerisce una disposizione gerarchica delle CA. Queste catene non sono eccessivamente lunghe (massimo 10 nodi), e per funzionare sfruttano due tipi di certificato:

  - *Forward certificates*: i certificati di X sono firmati da altre CA.
  - *Reverse certificates*: i certificati di altre CA sono firmati da X.

- *Rete di fiducia*
  Comprende modelli diretto e gerarchico, basandosi sull'idea che la fiducia dipende dal possessore del certificato e non è un valore assoluto. In questo modello i certificati sono spesso firmati da più entità, e se l'utente non ripone fiducia diretta a nessuna delle entità che lo ha firmato, può risalire la catena di fiducia per cercare entità fidate. Il *web of trust* è usato per i certificati PGP.

3.3.1 Pretty Good Privacy

I certificati del modello PGP possono avere molteplici firme, perfino il "self-signing". Ogni utente può creare o firmare certificati per altri utenti conosciuti, eliminando la dipendenza da un'infrastruttura centralizzata.

A differenza di X.509, la nozione di *trust* è esplicita nei certificati, e differenti livelli di fiducia permettono una gestione più complessa delle chiavi da parte degli utenti. Quando una chiave è stata compromessa, gli utenti (compreso il possessore del certificato) possono firmarla con trust a 0 per revocarla. PGP si basa quindi su un *reputation system* privo di strutture centralizzate, in cui la validità dei certificati dipende dalla fiducia nelle entità che li firmano.

4. Protocolli di sicurezza

L'obiettivo di questo capitolo è sfruttare diverse primitive di crittografia per ottenere dei protocolli di sicurezza che possano colmare i difetti delle singole tecniche studiate in precedenza.
Esempi:

- `RSA/Diffie-Hellman`
  Cifratura e distribuzione di chiavi su canali insicuri, **ma** richiedono autorizzazione.
- `Firme digitali`
  Garantiscono l'autorizzazione, **ma** non la timeliness del messaggio (possibili *replay-attack*).

Come possiamo creare una comunicazione sicura?

- Usare applicazioni sicure su canali insicuri (PGP, Kerberos, ...).
- Risolvere problemi di sicurezza su layer di rete inferiori.

Definiamo *protocollo* un insieme di regole che consente lo scambio di messaggi tra due o più principal. Un *protocollo crittografico* usa meccanismi crittografici per conseguire obiettivi di sicurezza.

4.1 Costruzione di un protocollo

Supponiamo di voler costruire da zero un protocollo di sicurezza per lo scambio di chiavi crittografiche. Per prima cosa dobbiamo definire uno *scenario* e gli *obiettivi* di sicurezza che vogliamo soddisfare.
`Scenario`:

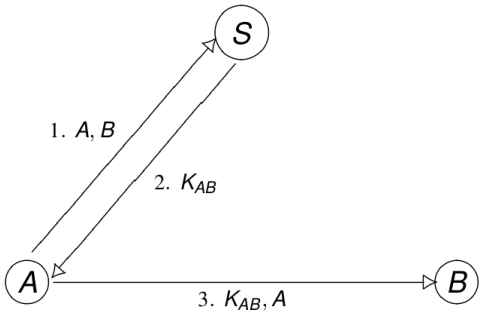
- Due utenti, A e B, intenzionati a comunicare.
- Un server S, dedito alla distribuzione di chiavi.

`Obiettivi`:

- Alla fine del protocollo la chiave K_{AB} dev'essere conosciuta solo da A, da B e opzionalmente da S.
- K_{AB} dev'essere **fresh**, ovvero creata appositamente per questa sessione.

Nella definizione del nostro protocollo consideriamo solo messaggi ben formati e che arrivano al destinatario, delegando ad altri livelli la gestione di ack o errori di trasmissione. Inoltre non vengono specificate le azioni interne dei principal (talvolta sono rilevanti per questioni di sicurezza, ma non è questo il caso).
NB: aggiungiamo una notazione. $\{M\}_K$ indica che il testo M è cifrato con la chiave pubblica K, mentre $[M]_K$ è firmato con la chiave segreta K.

4.1.1 Primo tentativo



Facciamo un primo tentativo, costruiamo un protocollo che consiste di tre passaggi:

1. A contatta S inviandogli l'identificativo delle due entità che intendono condividere la chiave.
2. S manda la chiave K_{AB} ad A.
3. A invia la chiave a B.

Una rappresentazione equivalente alla figura è la seguente:

1. $A \to S$: A, B
2. $S \to A$: K_{AB}
3. $A \to B$: K_{AB}, A

NB: "$A \to B$" serve a specificare sender e receiver, ma le identità A e B non sono parte del messaggio se non diversamente specificato.
In seguito utilizzaremo sempre questa nuova notazione.

Tornando al protocollo, notiamo un primo problema di notevole importanza: la chiave dev'essere comunicata solo ad A e B, nessun'altro deve poterla ricavare. Da ciò deriviamo la nostra prima *security assumption*:

> **Security assumption 1**
> L'avversario è capace di origliare tutti i messaggi scambiati in un protocollo di sicurezza.

4.1.2 Secondo tentativo

Sfruttiamo un algoritmo crittografico a chiave simmetrica per comunicare. Inizialmente S condivide con A la chiave K_{AS} e con B la chiave K_{BS}.

1. $A \to S$: A, B
2. $S \to A$: $\{K_{AB}\}_{K_{AS}}$, $\{K_{AB}\}_{K_{BS}}$
3. $A \to B$: $\{K_{AB}\}_{K_{BS}}$, A

In questo modo nessun avversario può leggere la chiave condivisa, ammesso

sia valida la seguente assunzione.

> **Perfect cryptography assumption**
> I messaggi cifrati possono essere letti solo dai legittimi destinatari, i quali sono in possesso della chiave di decifratura.

Tuttavia un avversario solitamente non è solo in grado di origliare i messaggi che passano sulla rete, bensì anche manipolarli.

> **Security assumption 2**
> L'avversario è capace di alterare tutti i messaggi usando qualsiasi informazione disponibile. Egli può inoltre reindirizzare messaggi ad altri principal (o semplicemente intercettarli) e generare messaggi completamente nuovi.

Possiamo riassumere le prime due assunzioni di sicurezza come segue:

> *L'avversario ha il completo controllo della rete.*

Vediamo un paio di possibili attacchi man-in-the-middle.

Man-in-the-middle tra A e B

1. $A \to S$: A, B
2. $S \to A$: $\{K_{AB}\}_{K_{AS}}$, $\{K_{AB}\}_{K_{BS}}$
3. $A \to C$: $\{K_{AB}\}_{K_{BS}}$, A
4. $C \to B$: $\{K_{AB}\}_{K_{BS}}$, D

L'avversario C intercetta il messaggio di A e lo manipola per convincere B che il mittente del messaggio sia D (dove D può essere chiunque, compreso C stesso). La chiave rimane segreta, tuttavia questo metodo può essere sfruttato per un *denial of service*.

Man-in-the-middle tra A ed S

1. $A \to C$: A, B
1′. $C \to S$: A, C
2. $S \to C$: $\{K_{AC}\}_{K_{AS}}$, $\{K_{AC}\}_{K_{CS}}$
2′. $C \to A$: $\{K_{AC}\}_{K_{AS}}$, $\{K_{AC}\}_{K_{CS}}$
3. $A \to C$: $\{K_{AC}\}_{K_{CS}}$, A

L'attaccante C altera il messaggio di A inviato ad S, chiedendo al server S di creare una chiave di sessione per A e C cifrata con chiave K_{CS}. A non può sapere se la chiave ottenuta è lecita o compromessa, e B non rileva alcuna anomalia in quanto C intercetta il messaggio originariamente rivolto a lui.

Questo è un attacco molto più pericoloso del precedente, in quanto C ora conosce la chiave e si è mascherato da B. Per funzionare tuttavia è necessario assumere che C sia un utente legittimo riconosciuto da S.

> **Security assumption 3**
> L'avversario può essere un utente legittimo (*insider*), un entità esterna (*outsider*) o una combinazione dei due.

4.1.3 Terzo tentativo

Per impedire l'attacco di cui sopra, alleghiamo crittograficamente le identità dei principals alla chiave.

1. $A \to S$: A, B
2. $S \to A$: $\{K_{AB},\, B\}_{K_{AS}},\ \{K_{AB},\, A\}_{K_{BS}}$
3. $A \to B$: $\{K_{AB}\}_{K_{BS}},\ A$

A questo punto l'attaccante non può compromettere il protocollo mediante intercettazione o alterazione di messaggi, a patto che A e B siano onesti.

Il protocollo non può ancora dirsi sicuro, infatti c'è una vulnerabilità che nasce dalla differenza di qualità tra le chiavi a lungo termine (scambiate con S) e le chiavi di sessione. Quest'ultime sono generate al fine di ottenere prestazioni migliori durante la comunicazione, ma aprono la strada a nuovi possibili attacchi: *replay attack*.

> **Security assumption 4**
> L'avversario è capace di ricavare la chiave di sessione K_{AB} usata in scambi di messaggi "sufficientemente vecchi".

Replay attack

1. $A \to C$: A, B
2. $C \to A$: $\{K'_{AB}\}_{K_{AS}},\ \{K'_{AB}\}_{K_{BS}}$
3. $A \to B$: $\{K'_{AB}\}_{K_{BS}},\ A$

L'attaccante intercetta il messaggio di A rivolto al server S, il quale non ha alcun ruolo in questo attacco. Nelle veci del server, C invia ad A una chiave di sessione K'_{AB} già usata in passato, perciò già ricavata da C.

Alla fine del protocollo, A e B condivideranno la chiave compromessa K'_{AB}, perciò C potrà leggere ed eventualmente modificare qualcunque messaggio scambiato tra i due principals. *Come possiamo prevenirlo?*

Metodo del **challenge-response**: uno dei principal genera e invia un *nonce* ("number used only once"), ovvero sceglie un numero casualmente. Quest'ultimo viene ritornato al mittente per verificare che il messaggio sia generato ex-novo.

4.1.4 Protocollo di Needham-Schroeder

1. $A \to S$: A, B, N_A
2. $S \to A$: $\{K_{AB}, B, N_A, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$
3. $A \to B$: $\{K_{AB}, A\}_{K_{BS}}$
4. $B \to A$: $\{N_B\}_{K_{AB}}$
5. $A \to B$: $\{N_B - 1\}_{K_{AB}}$

Nel protocollo di Needham-Schroeder, il nonce N_A inviato da A e ritornato da S permette di assicurare non solo la *freshness* del pacchetto per A, ma anche quella del pacchetto per B siccome è cifrato nello stesso messaggio. Non avremmo ottenuto le stesse proprietà di sicurezza se il secondo messaggio fosse stato il seguente:

2. $S \to A$: $\{K_{AB}, B, N_A\}_{K_{AS}}$, $\{K_{AB}, A\}_{K_{BS}}$

I messaggi 4 e 5 consistono in un ulteriore handshake tra A e B per assicurare il secondo principal della freshness della chiave K_{AB}.
NB: *i nonce sono semplici numeri*, non contengono indizi riguardo l'identità del principal che li ha generati.

Il protocollo di Needham-Schroeder è vulnerabile ad un attacco man-in-the-middle conosciuto come attacco di *Denning and Sacco*.

Denning and Sacco attack

Questo attacco man-in-the-middle si basa su una vulnerabilità del protocollo Needham-Schroeder, secondo il quale nessuno a parte A sia in grado di rispondere alla challenge di B del quarto messaggio. Per la quarta security assumption, sappiamo che un avversario C potrebbe conoscere una vecchia chiave di sessione K'_{AB}. C potrebbe dunque fingersi A e persuadere B a usare la chiave di sessione di cui è a conoscenza.

3. $C \to B$: $\{K'_{AB}, A\}_{K_{BS}}$
4. $B \to C$: $\{N_B\}_{K'_{AB}}$
5. $C \to B$: $\{N_B - 1\}_{K'_{AB}}$

4.1.5 Quinto (e ultimo) tentativo

Affinché il protocollo non sia vulnerabile ai replay attack, sia A che B devono inviare una challenge a S.

1. $B \to A$: B, N_B
2. $A \to S$: A, B, N_A, N_B
3. $S \to A$: $\{K_{AB}, B, N_A\}_{K_{AS}}$, $\{K_{AB}, A, N_B\}_{K_{BS}}$
4. $A \to B$: $\{K_{AB}, A, N_B\}_{K_{BS}}$

Questa volta la comunicazione viene iniziata da B, il quale invia il proprio nonce ad A. Quest'ultimo comunica con il server che gli risponde con le chiavi di sessione separate, ognuna contenuta nella rispettiva response per garantire la freshness.

Può sembrare di aver ottenuto più proprietà con meno scambi di messaggi, a differenza del quarto tentativo abbiamo in realtà perso la proprietà di *key confirmation*: al termine del protocollo non sappiamo con certezza se B è a conoscenza della chiave.

La versione finale di questo protocollo non è vulnerabile ad alcun attacco tra quelli visti fino ad ora, tuttavia è presto per poter affermare che il protocollo è *sicuro*, dobbiamo prima dare una definizione formale degli obiettivi che deve soddisfare.

4.1.6 Security assumptions

Riassumiamo le assunzioni di sicurezza:

0. (*Perfect cryptography assumption*) I messaggi cifrati possono essere letti solo dai legittimi destinatari, i quali sono in possesso della chiave di decifratura.
1. L'avversario è capace di origliare tutti i messaggi scambiati in un protocollo di sicurezza.
2. L'avversario è capace di alterare tutti i messaggi usando qualsiasi informazione disponibile. Egli può inoltre reindirizzare messaggi ad altri principal (o semplicemente intercettarli) e generare messaggi completamente nuovi.
3. L'avversario può essere un utente legittimo (*insider*), un entità esterna (*outsider*) o una combinazione dei due.
4. L'avversario è capace di ricavare la chiave di sessione K_{AB} usata in scambi di messaggi "sufficientemente vecchi".

4.2 Needham-Schroeder Public Key

Studiamo il protocollo Needham-Schroeder Public Key (NSPK), che mira a garantire l'autenticazione dei principal:

1. $A \to B$: $\{N_A, A\}_{K_B}$
2. $B \to A$: $\{N_A, N_B\}_{K_A}$
3. $A \to B$: $\{N_B\}_{K_B}$

NB: è omessa l'interazione con il server per chiedere le chiavi pubbliche.
Obiettivi del protocollo:

- *Autenticazione* dei messaggi.
- *Timeliness*, ovvero freshness dei messaggi.
- *Segretezza*, se fossero violati i nonce potrebbe fallire l'autenticazione.

Modello dell'attaccante: questi conosce il protocollo ma non può rompere la crittografia, inoltre può essere *attivo* o *passivo* se rispettivamente interviene nella conversazione o se ascolta solamente.

Dolev-Yao attacker: modello standard di attaccante, molto potente in quanto ha il pieno controllo della rete, senza tuttavia invalidare l'assunzione di perfect cryptography.

4.2.1 Lowe attack

L'attacco di Lowe a NSPK è un man-in-the-middle in cui A vuole comunicare con C, ma quest'ultimo ha intenzioni malevole e vuole fingersi A per comunicare con B.

1. $A \to C$: $\{N_A, A\}_{K_C}$
2. $C \to B$: $\{N_A, A\}_{K_B}$
3. $B \to C$: $\{N_A, N_B\}_{K_A}$
4. $C \to A$: $\{N_A, N_B\}_{K_A}$
5. $A \to C$: $\{N_B\}_{K_C}$
6. $C \to B$: $\{N_B\}_{K_B}$

Protocollo Needham-Schroeder-Lowe (NSL): nella risposta B invia la propria identità, oltre ai nonce.

1. $A \to B$: $\{N_A, A\}_{K_B}$
2. $B \to A$: $\{N_A, N_B, B\}_{K_A}$
3. $A \to B$: $\{N_B\}_{K_B}$

4.2.2 Type-flaw attack

In un type-flaw attack un principal viene ingannato riguardo il tipo di dato che riceve: l'attaccante invia al principal un messaggio con il giusto numero di bit, ma il cui contenuto è di tipo diverso. Questo attacco è inoltre un *oracle attack*, in quanto la vittima, seguendo il protocollo, invia all'attaccante informazioni utili per proseguire nell'attacco.

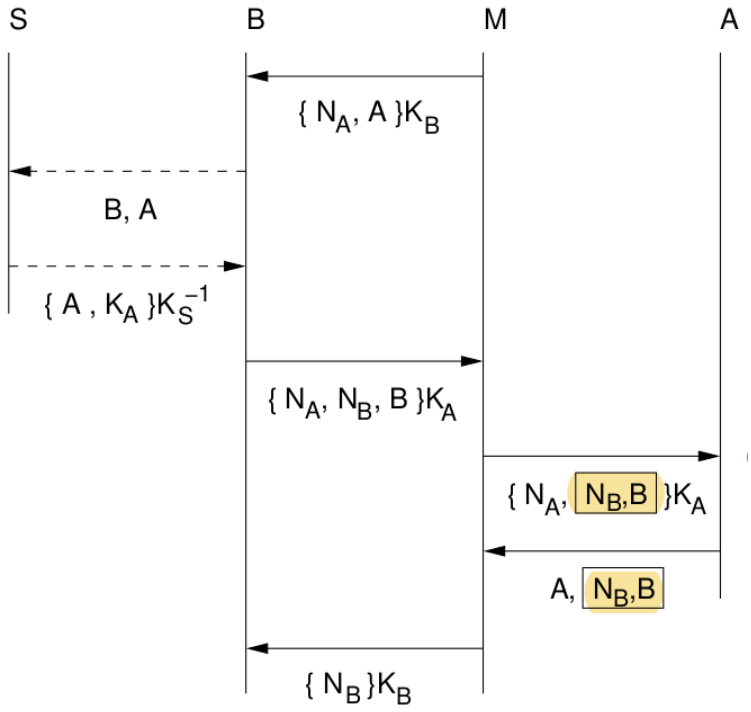


Figura 4.1: Oracle attack da parte dell'attaccante M.

In figura [4.1] vediamo come l'attaccante M (man-in-the-middle) intenda fingersi A nella comunicazione con B. Per riuscire nel suo intento, M manda a B il primo messaggio del protocollo NSL indicando A come mittente. Quando B risponde con un messaggio cifrato con la chiave pubblica di A, M lo rigira ad A come fosse il primo messaggio del protocollo. A quindi interpreterà (erroneamente) il tipo del messaggio e cercherà di contattare S chiedendo una chiave per comunicare con "N_B, B". M intercetterà quest'ultimo messaggio e potrà rispondere infine a B per aprire una comunicazione fingendosi A.
Versione definitiva, che fa uso di firme digitali per evitare type-flaw attack:

1. $A \to B$: $\{[N_A, A]_{K_A^{-1}}\}_{K_B}$
2. $B \to A$: $\{N_A, [N_B]_{K_B^{-1}}\}_{K_A}$
3. $A \to B$: $\{[N_B]_{K_A^{-1}}\}_{K_B}$

4.3 Tipi di attacco

Elenchiamo i principali tipi di attacco, per poi vedere esempi di protocolli con relative vulnerabilità:

1. *Man-in-the-middle*
   L'attaccante si posiziona in mezzo a due principal, intercettandone i messaggi per invalidare il protocollo: $A \leftrightarrow Z \leftrightarrow B$.
2. *Parallel session*
   Man-in-the-middle in cui l'attaccante sfrutta più run del protocollo contemporaneamente.
3. *Replay*
   L'attaccante riusa parti di messaggi precedenti come fossero fresh.
4. *Masquerading*
   L'attaccante finge di essere un altro principal.
5. *Reflection*
   Alcune informazioni vengono ritrasmesse al mittente.
6. *Oracle*
   Istanziando un protocollo con terzi principal, l'avversario può sfruttare le risposte legittime al protocollo per fare encryption/decryption di messaggi.
7. *Type-flaw*
   L'attaccante sostituisce campi del messaggio con informazioni di altro genere, senza invalidare la formattazione del messaggio stesso.

4.3.1 Otway-Rees protocol

Protocollo server-based per la distribuzione autenticata di chiavi, che garantisce segretezza e freshness delle chiavi, senza però assicurare autenticazione dei principal e key-confirmation.

1. $A \to B$: $I, A, B, \{N_A, I, A, B\}_{K_{AS}}$
2. $B \to S$: $I, A, B, \{N_A, I, A, B\}_{K_{AS}}, \{N_B, I, A, B\}_{K_{BS}}$
3. $S \to B$: $I, \{N_A, K_{AB}\}_{K_{AS}}, \{N_B, K_{AB}\}_{K_{BS}}$
4. $B \to A$: $I, \{N_A, K_{AB}\}_{K_{AS}}$

In questo caso, I è l'identificativo della run del protocollo.
Vediamo due possibili vulnerabilità del protocollo Otway-Rees:

- *Reflection/Type-flaw*
  Supponendo $|\{I, A, B\} = \{K_{AB}\}|$, un attaccante Z potrebbe intercettare il primo messaggio e, fingendosi B, rispondere ad A con $I, \{N_A, I, A, B\}_{K_{AS}}$.

A vede che il nonce è corretto, perciò accetta $\{I, A, B\}$ come chiave di sessione.

- In modo analogo al precedente attacco, Z si maschera da server S e risponde a B "riflettendo" le parti cifrate dei primi messaggi.

  3. $Z \to B$: $I, \{N_A, I, A, B\}_{K_{AS}}, \{N_B, I, A, B\}_{K_{BS}}$

  In questo modo Z può leggere e manipolare la comunicazione tra A e B, invalidando key-authentication e segretezza.

4.3.2 Andrew Secure RCP protocol

In questo protocollo, due principal che condividono una chiave simmetrica K_{AB} intendono scambiarsi una nuova chiave condivisa K'_{AB} che sia autenticata, fresh e segreta.

1. $A \to B$: $A, \{N_A\}_{K_{AB}}$
2. $B \to A$: $\{N_A + 1, N_B\}_{K_{AB}}$
3. $A \to B$: $\{N_B + 1\}_{K_{AB}}$
4. $B \to A$: $\{K'_{AB}, N'_B\}_{K_{AB}}$

Type-flaw attack Un attaccante Z può fare man-in-the-middle fingendosi B, intercettando il terzo messaggio e rimandando ad A il secondo come quarto messaggio:

4. $Z \to A$: $\{N_A + 1, N_B\}_{K_{AB}}$

Al termine dell'attacco A accetta come nuova chiave di sessione $N_A + 1$, che non è autenticata. In questo caso la secrecy non è stata violata.

4.3.3 Key exchange with CA

Questo protocollo, conosciuto anche come "Denning & Sacco", permette a due principal in possesso di una chiave di sessione di scambiarsi il certificati di autenticazione. Aggiungiamo alcuni elementi alla notazione: C_A indica il certificato di A, T_A indica un timestamp generato da A. In questo caso inoltre S rappresenta il server di una CA.

1. $A \to S$: A, B
2. $S \to A$: C_A, C_B
3. $A \to B$: $C_A, C_B, \{\{T_A, K_{AB}\}_{K_A^{-1}}\}_{K_B}$

Un attacco a questo protocollo si presenta quando A vuole comunicare con Z, ma Z usa il messaggio ricevuto per contattare B fingendosi A:

$A \to Z$: $C_A, C_Z, \{\{T_A, K_{AZ}\}_{K_A^{-1}}\}_{K_Z}$

$Z \to B$: $C_A, C_B, \{\{T_A, K_{AZ}\}_{K_A^{-1}}\}_{K_B}$

Giunti a questo punto, B crede di comunicare con A, invece parla con Z: sono state invalidate sia l'autenticazione che la segretezza. Come difendersi? Nell'ultimo messaggio vanno specificati anche i principal interessati:

3. $A \to B$: $C_A, C_B, \{\{A, B, T_A, K_{AB}\}_{K_A^{-1}}\}_{K_B}$

4.3.4 Esempi di attacco

Vediamo qualche esempio di attacco: binding, parallel session e replay.

Binding attack

Il seguente protocollo:

1. $A \to S$: A, B, N_A
2. $S \to A$: $S, \{S, A, N_A, K_B\}_{K_S^{-1}}$

ammette un *binding attack*:

1.1. $A \to Z$: A, B, N_A
2.1. $Z \to S$: A, Z, N_A
2.2. $S \to Z$: $S, \{S, A, N_A, K_Z\}_{K_S^{-1}}$
1.2. $Z \to A$: $S, \{S, A, N_A, K_Z\}_{K_S^{-1}}$

In questo caso Z si spaccia per B nella comunicazione.
Fix: aggiungiamo l'identità del destinatario B nel secondo messaggio.

2. $S \to A$: $S, \{S, A, N_A, B, K_B\}_{K_S^{-1}}$

Parallel session attack

Il seguente protocollo one-way di autenticazione:

1. $A \to B$: $\{N_A\}_{K_{AB}}$
2. $B \to A$: $\{N_A + 1\}_{K_{AB}}$

ammette un *parallel-session attack* con "oracle":

1.1. $A \to Z$: $\{N_A\}_{K_{AB}}$
2.1. $Z \to B$: $\{N_A\}_{K_{AB}}$
2.2. $A \to Z$: $\{N_A + 1\}_{K_{AB}}$
1.2. $Z \to A$: $\{N_A + 1\}_{K_{AB}}$

A è costretto ad agire da oracolo contro se stesso, fornendo a Z la risposta alla propria stessa challenge. L'attaccante quindi, sfruttando due sessioni dello stesso protocollo, riesce ad autenticarsi mascherato da B.
Fix: aggiungiamo l'identità del mittente A al primo messaggio.

1. $A \to B$: $\{N_A, A\}_{K_{AB}}$

Replay attack

Il protocollo Needham-Schroeder Shared Key ammette un *replay attack*, vedi sezione [4.1.4].

4.3.5 Buone norme per protocolli di sicurezza

- Ogni messaggio deve contenere tutto il necessario (incluse le identità dei principal).
- Dev'essere chiaro perché determinati meccanismi crittografici vengono applicati.
- Dev'essere chiaro perché vengono sfruttati i nonce.
- Tener conto di possibili replay-attack approfittando di challenge-response.
- Devono essere chiare le relazioni di fiducia tra i principal.

5. Protocolli di sicurezza moderni

5.1 Kerberos

Protocollo per l'autenticazione ed il controllo degli accessi in ambienti aperti e distribuiti. Il nome del protocollo è dovuto all'intezione di implementare una proprietà per ogni testa di Cerbero: autenticazione, accounting e auditing. È stata realizzata solo la prima.
Caratteristiche di Kerberos:

- *Sicuro*
  Nessun attaccante può impersonare un utente del sistema.
- *Affidabile*
  I servizi offerti da Kerberos devono essere sempre fruibili da tutti i programmi dell'architettura distribuita.
- *Trasparente*
  Deve prevedere un meccanismo di *single sign-on* (SSO), affinché l'utente possa usare un'unica password per tutti i servizi offerti.
- *Scalabile*
  Il sistema dev'essere scalabile per supportare un vasto numero di utenti.

L'architettura di Kerberos si compone di due server: il *Kerberos Authentication Server* (KAS) ed il *Ticket Granting Server* (TGS). Inoltre è presente un controllo degli accessi per controllare i ticket emessi dal TGS.

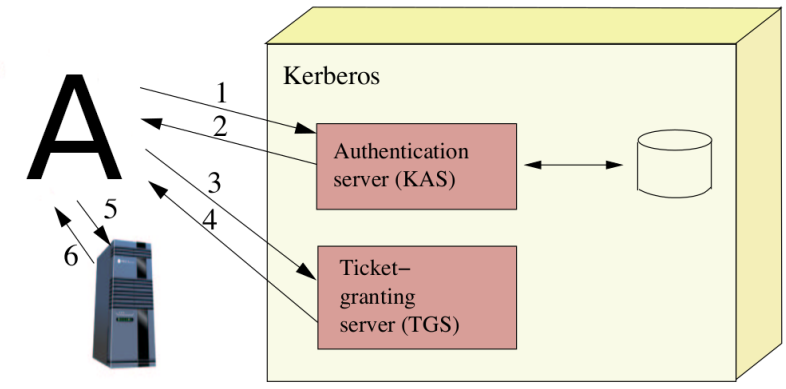


Figura 5.1: Architettura di Kerberos.

5.1.1 Funzionamento

Il funzionamento di Kerberos si divide in tre fasi basate su Needham-Schroeder, usando tuttavia i timestamps al posto dei nonce per garantire freshness.

- Autenticazione dell'utente.
- Autorizzazione per l'uso di un certo servizio.
- Connessione al servizio.

La fase di autenticazione viene svolta al login, in seguito alla quale l'utente è autenticato per svariate ore (la durata dipende all'applicazione):

1. $A \to KAS$: A, TGS
2. $KAS \to A$: $\{K_{A,TGS}, TGS, T_1, \underbrace{\{A, TGS, K_{A,TGS}, T_1\}_{K_{KAS,TGS}}}_{\text{Authentication Ticket}}\}_{K_{AS}}$

L'utente A vuole eseguire il login, perciò KAS accede al database e invia ad A una chiave di sessione $K_{A,TGS}$ ed un *Authentication Ticket* cifrato. La validità della chiave di sessione è legata al timestamp T_1. L'intero messaggio è cifrato con chiave K_{AS}, che deriva dalla password dell'utente.

Per accedere ad un servizio B, A deve presentare l'auth ticket ricevuto al login insieme ad un *Authenticator*:

3. $A \to TGS$: $\underbrace{\{A, TGS, K_{A,TGS}, T_1\}_{K_{KAS,TGS}}}_{\text{Auth Ticket}}, \underbrace{\{A, T_2\}_{K_{A,TGS}}}_{\text{Authenticator}}, B$
4. $TGS \to A$: $\{K_{A,B}, B, T_3, \underbrace{\{A, B, K_{A,B}, T_3\}_{K_{B,TGS}}}_{\text{Service Ticket}}\}_{K_{A,TGS}}$

Questa fase dev'essere svolta ogni volta che A vuole accedere ad un servizio B. Inoltre il timestamp T_2 ha un lifetime di pochi secondi per prevenire replay attacks; il server tiene traccia dei recenti authenticators per prevenire replay immediati. Nel quarto messaggio, il TGS invia ad A una nuova chiave di sessione $K_{A,B}$ con validità di pochi minuti ed un *Service Ticket*.

Nella terza ed ultima fase, A contatta il servizio B fornendo il service ticket ed un nuovo authenticator cifrato con la chiave di sessione:

5. $A \to B$: $\underbrace{\{A, B, K_{A,B}, T_3\}_{K_{B,TGS}}}_{\text{Service Ticket}}, \underbrace{\{A, T_4\}_{K_{A,B}}}_{\text{Authenticator}}$
6. $B \to A$: $\{T_4 + 1\}_{K_{A,B}}$

Il sesto messaggio (opzionale) conferma l'accesso di A al servizio B.

5.1.2 Scalabilità

Kerberos risulta altamente scalabile grazie alla suddivisione dei servizi in *reami*. Ogni reame è definito e gestito da un server Kerberos, il cui TGS

conosce i servizi del proprio reame e il TGS dei servizi esterni. Nel caso un utente A volesse accedere ad un servizio B di un altro reame, il TGS di A lo indirizzerebbe al server corretto fornendogli una chiave per contattarlo (fase 2 svolta due volte).

5.1.3 Limitazioni della versione IV

- Sfruttanto il primo messaggio, un attaccante potrebbe fare *flooding* contro il KAS, con conseguente DoS.
- La doppia cifratura nel secondo messaggio è ridondante (eliminata nella versione V).
- La difesa dai replay attack dipende eccessivamente dai timestamp, se l'utente è compromesso tali attacchi risultano possibili.

Alcune di queste limitazioni sono tuttora riscontrabili nella versione V.

5.2 TLS/SSL

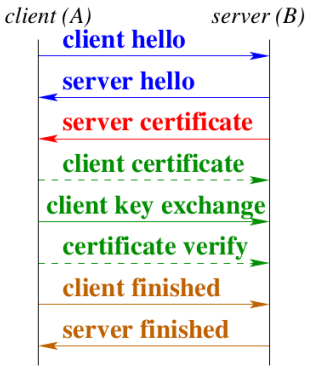
Secure Sockets Layer e *Transport Layer Security* sono protocolli di sicurezza per il livello di trasporto, non sono gestiti dal sistema operativo ma dall'applicazione. Sono lo stesso protocollo, ma sfruttano algoritmi crittografici diversi (TLS è successivo a SSL ed è stato standardizzato nel 1999). L'obiettivo di TSL/SSL è garantire privacy e integrità dei dati tra due applicazioni che comunicano (*end-to-end*), anche in presenza di attacchi via rete.

SSL consiste di due protocolli: uno di handshake e uno di record. Il primo utilizza la crittografia a chiave asimmetrica per stabilire diverse chiavi di sessione, il secondo sfrutta le chiavi generate per garantire confidenzialità, integrità e autenticità durante la comunicazione tra client e server.

5.2.1 Handshake

L'handshake prevede la negoziazione sulla versione del protocollo da usare e l'autenticazione facoltativa di client e server mediante certificati digitali. Il protocollo si suddivide in 4 fasi:

1. Negoziazione delle versioni del protocollo.
2. Scambio del certificato del server.
3. Scambio della chiave del client.
4. Instaurazione della connessione.



Fase 1. *Hello*

1. $A \rightarrow B$: A, N_A, SID, P_A
2. $B \rightarrow A$: N_B, SID, P_B

A identifica il client (in pratica, l'IP nel protocollo TCP). B identifica il server. SID è il *Session Identifier*. P_A indica le preferenze di A riguardo gli algoritmi di cifratura e compressione. Il server mette in P_B gli algoritmi più sicuri tra quelli supportati da A.

Fase 2. *Server certificate*

3. $B \rightarrow A$: $certificate(B, K_B)$

Il server B invia ad A il proprio certificato, in formato X.509v3, firmato da un entità trusted.

Fase 3. *Client exchange*

4. $A \rightarrow B$: $certificate(A, K_A)$
5. $A \rightarrow B$: $\{PMS\}_{K_B}$
6. $A \rightarrow B$: $\{hash(\dots)\}_{K_A^{-1}}$

PMS è il *pre-master secret*, usato in seguito per generare un *master secret* M:

$$M = PRF(PMS, N_A, N_B)$$

dove PRF è una Pseudo-Random Function (hash).
Il quarto ed il sesto messaggio sono opzionali e raramente richiesti, servono per autenticare il client. Nel senso A invia l'hash computato usando tutti i messaggi precedenti.

Fase 4. *Finish*

7. $A \rightarrow B$: $\{Finished\}_{clientK}$
8. $B \rightarrow A$: $\{Finished\}_{serverK}$

Finished è l'hash dei messaggi precedenti, usato per impedire *downgrading attack* sugli algoritmi di cifratura. *clientK* e *serverK* sono chiavi simmetriche generate da N_A, N_B ed M.

5.2.2 Gestione delle chiavi private

I metodi per mantenere le chiavi private protette nel server sono:

- Sfruttare hardware ad-hoc.
- Mantenere le chiavi cifrate (serve una password ad ogni reboot).
- Mantenere le chiavi in chiaro.

Ad oggi la metodologia più utilizzata è la terza.

5.3 IPsec

Il protocollo *IPsec* nasce in quanto il protocollo IP non prevede meccanismi di sicurezza. Esso fornisce un canale sicuro per ogni applicazione, garantendo confidenzialità (cifratura dei dati), integrità (checksum) e autenticazione (firme e certificati). Dev'essere installato sul sistema operativo, o sui security gateway per implementare *Virtual Private Networks* (VPNs).

Per evitare di rallentare inutilmente le comunicazioni, IPsec non viene usato nelle reti locali, ma solo nel trasporto di pacchetti via internet.

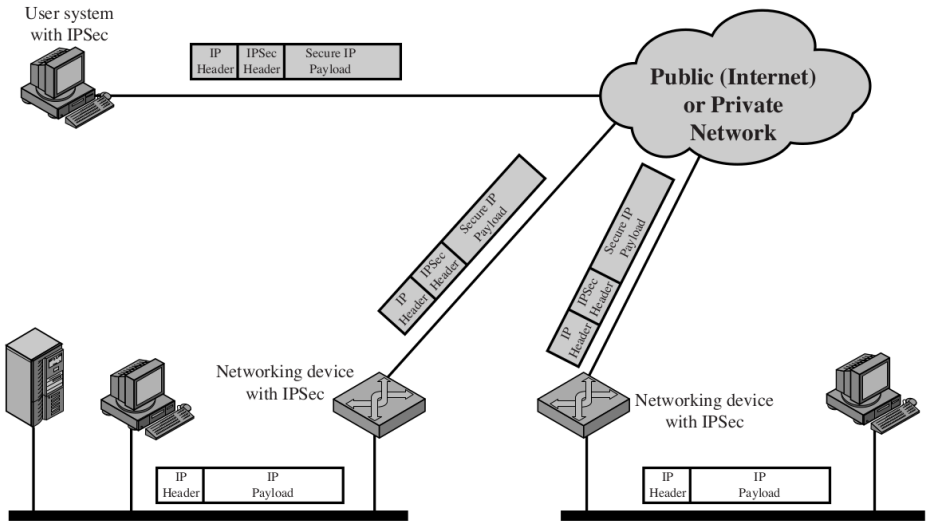


Figura 5.2: Un possibile scenario di IPsec.

IPsec si compone di tre diversi protocolli:

- *Authentication Header* (AH)
  Garantisce l'autenticazione (in seguito inglobato da ESP).
- *Encapsulatin Secuity Payload* (ESP)
  Garantisce confidenzialità e integrità.
- *Key Management* (IKE)
  Gestisce lo scambio di chiavi.

IPsec prevede due modalità di utilizzo: *Tunnel mode* e *Transport mode*. Nella prima modalità il pacchetto originale viene completamente cifrato e inserito in un nuovo pacchetto IP, il quale ha un nuovo header IP e un header IPsec. Tale meccanismo è comunemente utilizzato nelle VPN. Nella transport mode viene cifrato il payload e aggiunto un header IPsec, mantenendo l'originale header IP (vedi fig. [5.3]).

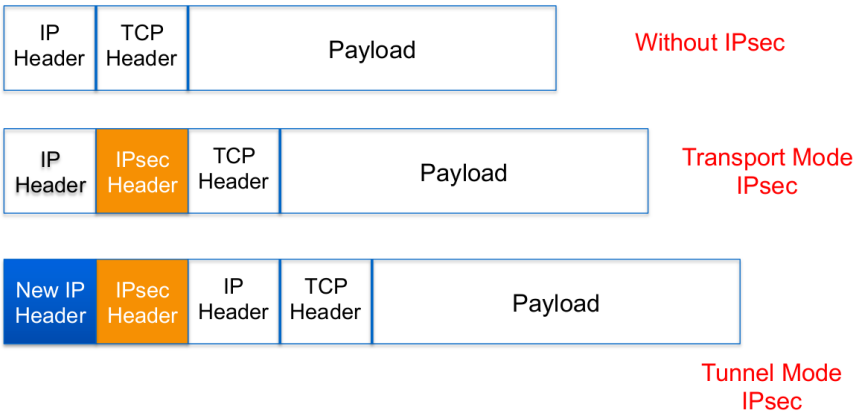


Figura 5.3: Tunnel mode e Transport mode di IPsec.

5.3.1 Authentication Header

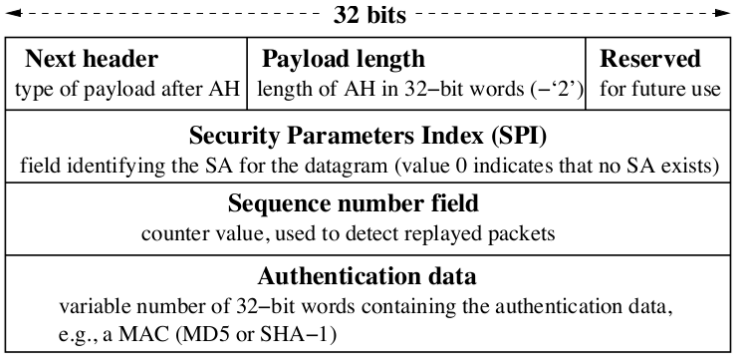


Figura 5.4: Authentication Header (AH).

L'header di IPsec si colloca tra l'header IP ed il layer di trasporto, contiene un *sequence number* per evitare replay attack e un campo relativo all'*authentication data* che identifica la comunicazione. Per poter funzionare inoltre, IPsec genera mediante IKE delle *Security Associations* (SA), le quali sono relazioni one-way tra mittente e destinatario. Una SA permette di specificare algoritmi per autenticazione e cifratura, chiavi, lifetime delle chiavi, protocol-mode (tunnel/transport). L'identificativo della SA è contenuto nel campo *Security Parameters Index* (SPI) dell'AH.

5.3.2 Encapsulating Security Payload

Il protocollo ESP si occupa di cifrare e, opzionalmente, autenticare il messaggio. Per garantire confidenzialità, cifra il payload e aggiunge in testa un

header contenente le informazioni per la corretta decifratura. L'autenticazione è ottenuta invece aggiungendo in coda al pacchetto un MAC ottenuto dall'hash dei precedenti campi.

Usando sia AH che ESP enll'header IPsec possiamo garantire confidenzialità, integrità, autenticazione e difesa da replay attack.

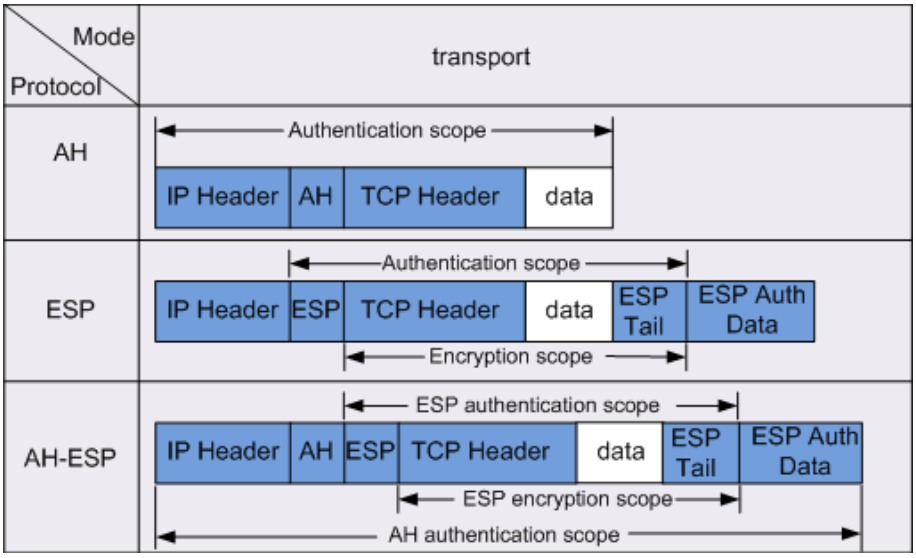


Figura 5.5: IPsec header

5.3.3 Internet Key Exchange

La gestione delle chiavi è delegata ad IKE, un protocollo flessibile ma complesso che permette di stabilire SA. IKE nasce come evoluzione di diversi protocolli basati su Diffie-Hellman, al quale aggiunge l'autenticazione mantenendo la *Perfect Forward Secrecy* (PFS), proprietà che garantisce la segretezza della sessione anche dopo la rottura della chiave long-term.

Il protocollo prevede due fasi, dette *Main mode* e *Quick mode*, di cui la prima è usata solo nel caso i due principal non si conoscano. Nella main viene stabilita la SA per la fase successiva, durante la quale vengono create delle SAs figlie per cifrare e autenticare le future comunicazioni.
Main mode:

1. $A \to B$: C_A, ISA_A
2. $B \to A$: C_A, C_B, ISA_B
3. $A \to B$: $C_A, C_B,\ X, g, p, N_A$
4. $B \to A$: C_A, C_B, Y, N_B
5. $A \to B$: $C_A, C_B, \{ID_A, AUTH_A\}_K$
6. $A \to B$: $C_A, C_B, \{ID_B, AUTH_B\}_K$

in cui:

- ISA: contiene una serie di proposte crittografiche
- X e Y sono le mezze chiavi: $X = g^x mod(p)$, $Y = g^y mod(p)$
- $K = Y^x mod(p) = X^y mod(p)$
- ID: identifica l'utente (IP, nome del dominio, certificato o email)
- $AUTH$: MAC o firma basato che autentica i messaggi precedenti.

Quick mode:

1. $A \rightarrow B$: $C_A, C_B, HASH(1),\ SA_A, N_A,\ X, g, p,\ ID_A, ID_B$
2. $B \rightarrow A$: $C_A, C_B, HASH(2),\ SA_B, N_B,\ Y, g, p,\ ID_A, ID_B$
3. $A \rightarrow B$: $C_A, C_B, HASH(3)$

in cui:

- $HASH(1) = h(SKEYID_a, \{SA_A, N_A, X, ID_A, ID_B\})$
- $HASH(2) = h(SKEYID_a, \{SA_B, N_A, N_B, Y, ID_A, ID_B\})$
- $HASH(3) = h(SKEYID_a, \{0, N_A, N_B\})$

dove $SKEYID_\alpha$ con $\alpha = a, e$ è una chiave derivata dalla chiave conosciuta nella prima fase, il cui pedice ne indica l'utilizzo (autenticazione o cifratura). Grazie alla PFS, anche se in futuro viene scoperta la chiave K stabilita nella main mode, le chiavi di sessione derivate non sono compromesse.

A. Teoria dei numeri

Blablabla