

Capitolo 1

Introduzione alla sicurezza

1.1 Il Morris Internet Worm

Il primo **worm** lanciato in rete è stato il **Morris Internet Worm**, dal nome del suo creatore *Tappan Morris*. Questo primo esempio di attacco informatico è datato 2 novembre **1988** ed è stato lanciato da un laboratorio del MIT.

Questo worm colpiva sistemi UNIX sfruttando delle vulnerabilità di alcuni software quali ad esempio **sendmail** (in particolare la sua modalità *debug*), **finger**, **rsh/rexec** e si manifestava tramite la comparsa di file strani in `/usr/tmp` e di strani messaggi nei log. Il tipo di attacco perpetrato ricade nella categoria dei **DOS** (*Denial Of Service*, argomento che approfondiremo in seguito). Il DOS si manifestava grazie al fatto che il worm si auto-reinstallava sullo stesso computer più volte, causandone il rallentamento e successivamente dei crash. Chiaramente il worm aveva fra le sue capacità anche quella di replicarsi fra più computer ed è ciò che ha portato a infettare oltre 6.000 computer.

Nel 1988 lo scenario di Internet era molto diverso da quello attuale. **Internet** era nata appena 5 anni prima dalla progenitrice **Arpanet**. Lo step decisivo nella nascita di Internet è stato l'adozione da parte di Arpanet del **TCP/IP**. Il 1988 è anche l'anno della nascita del **GARR** (*Gruppo per l'Armonizzazione delle Reti della Ricerca*), la rete universitaria italiana. Dalla nascita di Internet ad oggi il trend di crescita è impressionante ed ha visto crescere esponenzialmente ogni anno il numero di host connessi alla rete.

Oggi dunque, proprio in virtù del grandissimo numero di host connessi alla rete, la sicurezza assume un ruolo cruciale. È necessario investire sulla

sicurezza, rendere sicuri i propri apparati **prima** che gli attacchi possano essere perpetrati.

1.2 I tipi di attacchi informatici

Vediamo ora in quali categorie ricadono gli attacchi informatici:

- **Spoofing:** consiste nel fingersi qualcun'altro. Lo spoofing più comune è l'**IP Spoofing** in cui si falsifica il campo *Source Address* del pacchetto IP. In questo modo è possibile apparire con un IP diverso da quello reale.
- **Denial Of Service:** il *DOS* consiste nel negare l'accesso ad un servizio. Ad esempio il DOS può essere causato con la tecnica del **SYN flood**, ovvero con l'inondazione di pacchetti SYN. Quand'è che avviene una inondazione di pacchetti SYN? I SYN vengono inviati da un host per instaurare una connessione TCP con un altro host. Per ogni SYN ricevuto l'host riserva dello spazio nella sua *cache*. Un'inondazione di SYN provoca l'esaurimento dello spazio nella cache del server e di conseguenza un crash ed una indisponibilità del servizio offerto.
- **Hijacking:** è la tecnica del dirottare il traffico destinato da un host A ad un host B in modo tale che passi attraverso un terzo host C. L'host attaccante (C) si pone nel mezzo reinviando ad A i pacchetti provenienti da B e viceversa. Questa tecnica è nota come **Man in the Middle**.
- **Codice maligno e trojan:** un trojan horse è un programma che in realtà contiene delle insidie, contiene del codice maligno che una volta eseguito infetta il computer e mette a rischio i dati in esso contenuti. Al contrario dei virus i Trojan Horse non si replicano automaticamente.
- **Accesso non autorizzato:** un attacco informatico è l'accesso a risorse alle quali non si è autorizzati ad accedere.
- **Port scanning/probing:** il port scanning/probing viene utilizzato per scoprire quali porte sono aperte e quali servizi sono in ascolto su di esse per poi perpetrare degli attacchi.
- **Virus e worm:** sappiamo benissimo in cosa consistono i virus e gli worm. Son dei programmi maligni che infettano particolari sistemi

operativi. I virus si differenziano dagli worm in quanto si attaccano ad un file ospite; gli worm al contrario sono programmini indipendenti.

- **Data sniffing:** il data sniffing altro non è che l'ascolto del canale per la cattura delle informazioni scambiate fra gli host vittime dell'attacco. Il data sniffing è ancora oggi molto produttivo data la grossa mole di dati che vengono inviati in chiaro.

1.3 Definizioni di base

Per introdurre il tema, è utile citare le tre proprietà di base per la sicurezza dell'informazione.

- **Confidenzialità:** la confidenzialità mira a garantire che, nello scambio di informazioni fra un mittente ed un destinatario, un terzo soggetto non possa venire a conoscenza del contenuto di tali risorse.
- **Integrità:** l'integrità è la caratteristica che possiede un dato che non ha subito modificazioni nel contenuto (intenzionali o accidentali).
- **Disponibilità:** nell'ambito della sicurezza dell'informazione, assicurare la disponibilità dei dati significa prevenire la non accessibilità delle informazioni ai legittimi utilizzatori.

Quanto miriamo a proteggere delle informazioni, siamo interessati a garantire il rispetto di una o più delle suddette caratteristiche. Inoltre vogliamo esser certi dell'identità dell'entità con cui interagiamo e questo ci porta a definire meccanismi di **autenticazione**. Il **non ripudio** di un'informazione è inoltre un'altro requisito che potremmo desiderare. Se un'informazione non è ripudiabile, vuol dire che l'autore non potrà mai disconoscerne la paternità.

Le informazioni possono essere a rischio nel momento in cui esiste nel sistema una **vulnerabilità**, ovvero un punto debole a fronte di una **minaccia** (una minaccia è circostanza che può potenzialmente creare danni). Mettere a segno un **attacco** significa sfruttare una vulnerabilità. Per evitare danni è bene adottare delle **contromisure** in grado di rimuovere le vulnerabilità.

1.4 Progettare la sicurezza

Nel proteggere le informazioni bisogna valutare alcuni aspetti. I principali emergono dalle seguenti domande.

- Quanto valgono le informazioni?
- Come possiamo quantificare il rischio di un attacco?
- Come possiamo quantificare il danno subito con la perdita di informazioni?
- Quanto costa proteggere le informazioni?

Questo vuol dire che nel progettare la sicurezza è importante valutare prima di tutto quanto valgono le informazioni. È bene individuare da cosa proteggere i dati (lettura da malintenzionati e/o alterazione e/o disponibilità) e quantificare i danni in caso di perdita. Quant'è probabile una perdita e quanto costa porre rimedio a delle vulnerabilità sono due aspetti importanti nel valutare il costo per proteggere le informazioni. Fare queste valutazioni è necessario in quanto, per proteggere dei dati, non possiamo spendere più di quanto perderemmo in caso di attacco.

Volendo progettare la sicurezza, le fasi da seguire prevedono:

- Analisi del contesto
- Analisi del sistema informatico
- Classificazione degli utenti
- Definizione dei diritti di accesso
- Catalogazione degli eventi indesiderati
- Valutazione del rischio
- Individuazione delle contromisure
- Integrazione delle contromisure

Per garantire che determinati requisiti di sicurezza siano rispettati, provvederemo a mettere in piedi dei **servizi di sicurezza** (*security service*). Un servizio di sicurezza fa uso di uno o più **meccanismi di sicurezza**, meccanismi destinati a rilevare, prevenire o recuperare da un attacco.

1.5 Modelli e attacchi

Nelle nostre trattazioni assumeremo sempre che sia presente un **canale insicuro**, mentre sorgente e destinazioni possono essere ritenuti sicuri.

Vediamo ora, in modo del tutto generale, come è possibile perpetrare attacchi ai requisiti di sicurezza indicati precedentemente.

- **attacco alla disponibilità:** gli attacchi alla disponibilità si effettuano normalmente con dei **DOS** (*Denial Of Service*). Si mira a negare l'accesso alle risorse a chi ne ha diritto, creando un'interruzione di servizio.
- **attacco alla confidenzialità:** l'attacco alla confidenzialità è, al contrario degli altri di cui abbiamo parlato o parleremo, un **attacco passivo** in quanto consiste semplicemente nell'intercettare il traffico per catturare le informazioni di interesse. Ovviamente in questo caso la contromisura più semplice è quella di cifrare il traffico, ma ne esistono altre (la cui praticabilità è da valutare volta per volta) come ad esempio l'utilizzo di un canale sicuro. Il controllo d'accesso al canale fa sì che sul canale non ci siano entità indesiderate.
- **attacco all'integrità:** l'attacco all'integrità prevede che le informazioni spedite da una sorgente S ad una destinazione D, vengano, durante il tragitto, modificate da un attaccante che cercherà (ovviamente) di fare in modo che D non sospetti nulla. Una contromisura consiste nel prevedere un meccanismo di accesso al canale e di avere degli attestati di integrità.
- **attacco all'autenticità:** un altro attacco attivo che prevede questa volta che l'attaccante crei dei messaggi e li invii ad una destinazione spacciandosi per un altro soggetto. Una contromisura consiste nel prevedere un meccanismo di accesso al canale e di avere degli attestati di integrità e d'origine.

Capitolo 2

Gestione della sicurezza informatica

2.1 Definizioni di base

Per garantire un adeguato livello di sicurezza informatica ad un'azienda è necessaria la stesura di un **piano di sicurezza** che tenga conto di un insieme di misure di carattere **tecnologico, organizzativo e normativo**. Il piano di sicurezza è normalmente redatto in due versioni: **pluriennale** (detto strategico e orientato al lungo periodo e poco dettagliato) e **annuale** (detto anche operativo e molto dettagliato).

Chiaramente non esiste un piano di sicurezza buono a priori per ogni azienda. Un piano di sicurezza è qualcosa che va realizzato ad-hoc per ogni situazione, pertanto la stesura di un piano è soprattutto un'opera di consulenza. Nonostante esistano differenti metodologie di stesura di un tale piano, tutte possono essere più o meno riconducibili alle fasi elencate qui di seguito:

- **Analisi del contesto**
- **Analisi del sistema informatico**
- **Analisi e valutazione dei rischi**
- **Individuazione e valutazione delle contromisure**
- **Formulazione del piano di sicurezza**

Adesso le analizziamo tutte nel dettaglio.

2.1.1 Analisi del contesto

Con questa fase si cerca di capire la natura dell'azienda, il contesto in cui opera, la sua organizzazione a livello di sedi, ruoli, competenze, responsabilità, processi... In questa fase si possono identificare due operazioni importanti: **rilevazione e documentazione del sistema informativo** e **analisi del contesto normativo e legislativo vigente**. Nella prima si individuano i flussi informativi (non necessariamente informatici), si individua in quali processi vengono utilizzati e quali requisiti di sicurezza sono richiesti per essi. Nella seconda si cerca invece di individuare i vincoli imposti dalla legge in materia di sicurezza, in modo tale da garantire l'adeguamento dell'azienda a tale contesto. Questa fase non richiede assolutamente conoscenze informatiche, ma anzi richiede più conoscenze organizzative.

2.1.2 Analisi del sistema informatico

Nella fase di analisi del sistema informatico si effettua un **censimento delle risorse hardware e software** al fine di individuare i punti di potenziale debolezza e le responsabilità di gestione di ogni componente. Particolarmente importante è il censimento di dati e archivi che prevede per ogni risorsa lo studio dei contesti operativi (quali procedure utilizzano dei dati, quali DBMS sono interessati in determinate operazioni ecc..), dei requisiti di sicurezza (classificazione dei diritti di accesso) e delle politiche di backup (modalità e frequenza di backup, tempi di conservazione..).

2.1.3 Analisi e valutazione dei rischi

Occorre valutare le vulnerabilità dell'azienda. È in questa fase che si raggruppano i dati in **data asset**, insiemi di dati creati secondo determinati criteri di utilizzo da parte di persone, processi ecc.. Ogni tipo di vulnerabilità può dar luogo ad un evento di attacco informatico. È importante notare che ogni dato viene protetto dalle vulnerabilità che vengono ritenute importanti, non da tutte in genere (ad esempio se si è interessati a mantenerne l'integrità si protegge solo quella, non si va a proteggere anche la segretezza se non necessario). Di tali eventi è necessario calcolare la probabilità di occorrenza e lo si fa basandosi su dati statistici, sui dati tecnici provenienti dagli apparati, sull'osservazione e l'analisi statistica, sull'esperienza personale. Per ogni tipo di vulnerabilità individuato si deve determinare il costo del danno arrecato tenendo presente sia **costi diretti** (costi di intervento, ripristino...) sia **costi indiretti** (danni di immagine, perdite finanziarie, violazione obblighi di legge). Il valore della perdita economica per ogni evento (che chiamia-

mo L) moltiplicato per la probabilità di occorrenza (P) determina la perdita annuale attesa o **ALE** (*Annual Loss Expectancy*): $ALE = L \cdot P$.

Esempi:

Evento: allagamento locale aziendale

Perdita presunta: 1.000.000 euro

Probabilità di occorrenza: 2% = 0,02

Perdita annuale attesa: $1.000.000 \cdot 0,02 = 20.000$

Evento: infezione di virus alle 300 postazioni aziendali

Perdita presunta per postazione: 125 euro

Probabilità di occorrenza per postazione: 200% = 2

Numero medio di PC interessati: 60% = 0,6

Perdita annuale attesa: $300 \cdot 0,6 \cdot 2 \cdot 125 = 45.000$

La metodologia appena vista è una metodologia di stima **quantitativa**, dove cioè si prova ad assegnare un valore numerico ad ogni evento e ad ogni probabilità. Questo metodo non è sempre applicabile. In alcuni casi non è possibile fare delle quantificazioni e quindi si ricorre all'analisi **qualitativa**. In questo tipo di analisi si effettua uno studio della terna asset-minaccia-vulnerabilità. Per ogni asset si studia la probabilità che si concretizzi una minaccia, e l'impatto che ciò avrebbe. Queste due grandezze si classificano fra livelli (come in genere alto, medio, basso) e l'incrocio delle due grandezze sul grafico ci restituisce il livello di rischio.

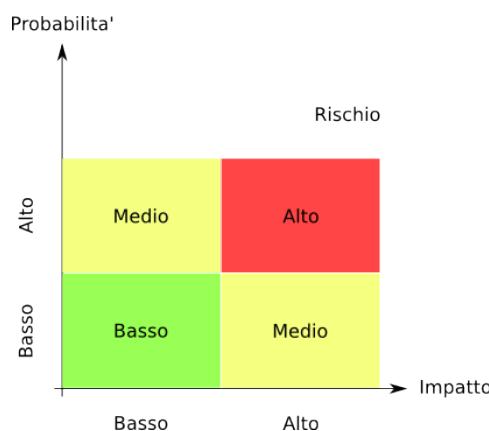


Figure 2.1: Esempio di analisi qualitativa

2.1.4 Individuazione e valutazione delle contromisure

In questa fase è necessario individuare e valutare le contromisure per le vulnerabilità individuate. Tali contromisure vanno valutate in termini di rapporto costo/efficienza. Una contromisura non può essere più costosa di quanto l'azienda perderebbe in caso di attacco.

Evento: infezione di virus alle 300 postazioni aziendali

Perdita presunta per postazione: 125 euro

Probabilità di occorrenza per postazione: $200\% = 2$

Numero medio di PC interessati: $60\% = 0,6$

Perdita annuale attesa: $ALE = 300 \cdot 0,6 \cdot 2 \cdot 125 = 45.000$

Contromisura: installazione di antivirus su tutti i computer

Costo antivirus per pc: 50 euro

Efficacia: $80\% = 0,8$

Prob. di occorrenza in presenza di contromisure: $2 - (1 - 0,8) = 0,4$

Perdita in presenza di contromisure: $ALEc = 300 \cdot 0,6 \cdot 0,4 \cdot 125 = 9000$

Risparmio da contromisure: $45.000 - (9000 + 50 \cdot 300) = 21.000$

Questa fase richiede conoscenze avanzate di informatica.

2.1.5 Formulazione del piano di sicurezza

A questo punto sulla base delle precedenti fasi si stende il piano di sicurezza aziendale che evidenzi cosa è necessario fare per proteggere l'azienda dalle vulnerabilità e prevede anche ogni quanto intervenire per la manutenzione.

Capitolo 3

Dati sicuri - parte I

3.1 Introduzione

La vasta scienza della **crittologia**, scienza che studia le scritture nascoste, ha due grandi branche: la **crittografia** e la **crittanalisi**. La crittografia studia metodi sempre più avanzati e sicuri per l'occultamento di determinati segni; la crittanalisi studia i metodi per decifrare testi occultati. L'una è la controparte dell'altra.

Perchè ci interessiamo di crittologia? Perchè le informazioni per essere ritenute al sicuro su un canale insicuro devono essere prima trasformate secondo opportuni algoritmi e protocolli (che impiegano codifiche ridondanti dei dati). Sebbene la crittografia esista da tempi antichi, solo al giorno d'oggi ha raggiunto un buon livello di complessità, grazie anche allo studio di materie come *Teoria degli algoritmi*, *Teoria dell'informazione*, *Teoria dei numeri* e *Teoria della probabilità*.

Fondamentale, nel momento in cui si voglia proteggere delle informazioni, è l'applicazione di trasformazioni segrete (dopo verrà spiegato meglio) e ottenute tramite calcoli impossibili (anche questo aspetto verrà approfondito in seguito). Un malintenzionato non deve avere alcuna possibilità di **sapere** quale trasformazione è stata applicata, nè deve poterlo **indovinare** o **dedurre**.

Parlavamo di calcoli impossibili. In realtà di impossibile non c'è nulla. Per il momento ci accontentiamo di dire che un attaccante deve compiere calcoli difficili per svelare un segreto, mentre dev'essere facile il calcolo necessario alla sorgente e alla destinazione per trasformare l'informazione in un senso e nell'altro. Possiamo dunque dire che ad azioni lecite devono corrispondere calcoli semplici, ad azioni illecite calcoli difficili.

Studiamo ora alcuni dei principali requisiti di sicurezza che si possono voler conseguire.

3.2 Integrità

Preservare l'integrità, come sappiamo, vuol dire proteggere il dato da alterazioni accidentali o intenzionali. Si utilizzano dei codici ridondanti per accertarsi dell'integrità dei dati.

Il caso più semplice è quello in cui la sorgente spedisca al destinatario, oltre al messaggio, anche un suo riassunto. Questo viene spedito su un canale sicuro per evitare alterazioni. Chiaramente se si dispone di un canale sicuro si potrebbe pensare di inviare direttamente il messaggio su di esso. Ciò nella realtà non sempre è fattibile in quanto i canali sicuri sono normalmente costosi.

Per proteggere l'informazione da errori casuali potrebbe bastare anche un semplice controllo di parità, o una checksum, o ancora un CRC (queste funzioni sono dette *funzioni hash semplici*). Ciò non è sufficiente per proteggere da alterazioni intenzionali, in quanto un malintenzionato potrebbe facilmente ricreare un messaggio avente lo stesso CRC. Ecco perciò che si rendono necessarie **funzioni hash crittografiche**. Queste funzioni producono un riassunto pressoché univoco e sarà difficile trovare un messaggio avente lo stesso hash. La destinazione calcolerà allo stesso modo il riassunto e, confrontandolo con quello ricevuto dalla sorgente, potrà eventualmente accorgersi di alterazioni.

Le funzioni hash hanno la caratteristica di produrre, dato un messaggio di lunghezza m , una stringa di lunghezza n , con $m > n$. Questo, com'è facile immaginare, può causare delle **collisioni**, ovvero può dare origine a riassunti uguali per più parole. Una buona funzione di hash deve minimizzare le probabilità di collisioni (dev'essere dunque **resistente**). Inoltre, per rendere il compito difficile all'attaccante, deve comportarsi come un oracolo casuale, cioè deve fare in modo che i simboli utilizzati siano equiprobabili. Una funzione di hash sicura non deve permettere di generare intenzionalmente collisioni, deve rendere computazionalmente impossibile tale operazione. Se così non fosse, un attaccante potrebbe alterare/sostituire il messaggio trasmesso dalla sorgente con un messaggio che restituisca la stessa checksum e la destinazione non se ne accorgerebbe.

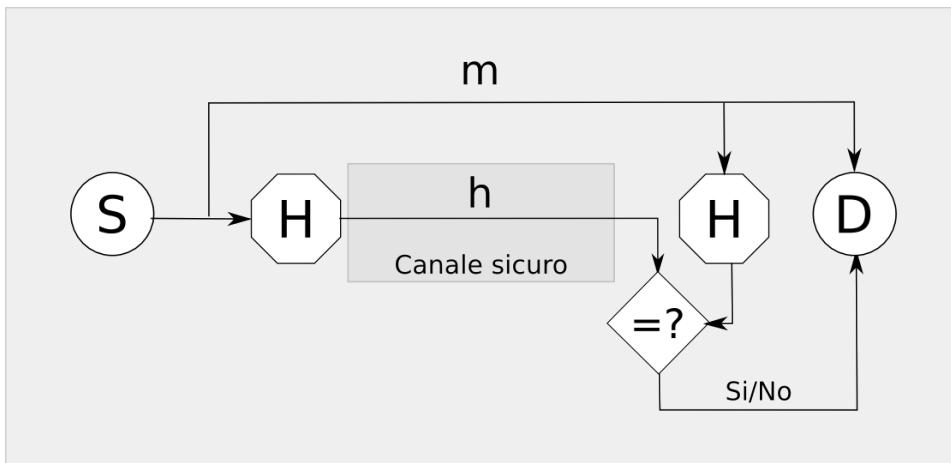


Figure 3.1: Integrità dei dati

Come possiamo vedere dalla figura 3.1, S invia un messaggio m a D . Su un canale sicuro invia il risultato dell'hash del messaggio (h) ottenuto tramite il blocco H . D quando riceve il messaggio ne calcola l'hash e lo confronta con quello ricevuto sul canale sicuro. L'eventuale intruso non modifica l'hash, ma cerca di inviare un messaggio che abbia lo stesso hash. Possibile facilmente con funzioni hash semplici apportare tali modifiche, quasi impossibile (ma dipende dalla bontà della funzione hash) riuscirci se si usa una funzione hash crittografica.

3.3 Riservatezza

Volendo proteggere la riservatezza delle informazioni che trasmettiamo, è necessario ricorrere a metodi di cifratura. La sorgente concorda dunque con la destinazione un metodo di rappresentazione che renda incomprensibile il messaggio ad un malintenzionato. Anche in questo caso i calcoli necessari a cifrare il messaggio da parte della sorgente, e decifrarlo da parte della destinazione devono essere facili, mentre quelli per decifrare il messaggio da parte di un malintenzionato devon esser complicati (computazionalmente impossibili).

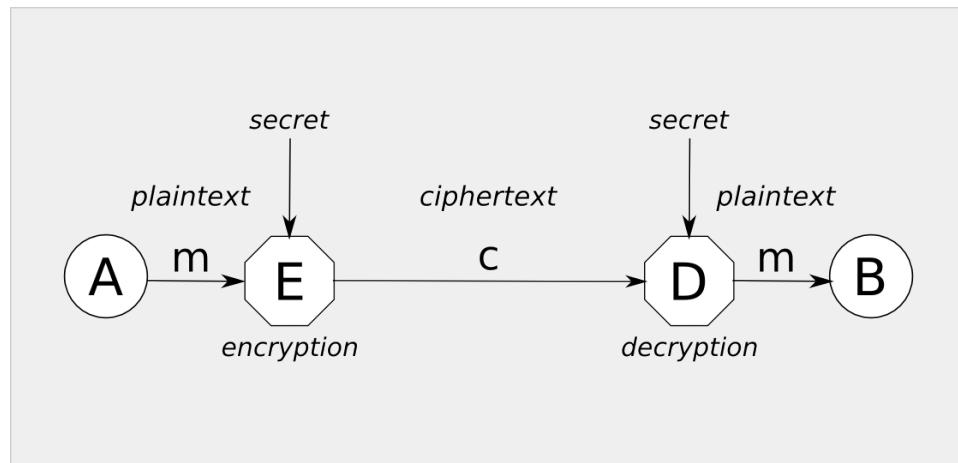


Figure 3.2: Riservatezza dei dati

In figura 3.2 possiamo osservare che il mittente A cifra il messaggio m usando un segreto concordato con il destinatario B . Il blocco E restituisce la versione criptata del messaggio m , ovvero il testo cifrato c . L'intrusore ascoltando il canale può solo ottenere c , dunque solo se è in possesso del segreto può comprendere cosa viene inviato. Il destinatario B , che invece conosce il segreto, può decifrare il testo tramite il blocco D .

Il protocollo utilizzato per difendere la riservatezza, ma contestualmente anche l'integrità è il seguente:

1. $p = m||H(m)$ - l'hash $H(m)$ viene concatenato al messaggio m
2. $c = E(p)$ - viene cifrato il testo p ottenuto al passo precedente (E sta per Encrypt)
3. $p^* = D(c^*) = m^*||H^*(m)$ - viene ricevuto il testo e decriptato (D sta per Decrypt)
4. $H^*(m) = ?H(m^*)$ - si controlla l'integrità

3.4 Autenticazione

Nel caso dell'autenticazione, la sorgente vuole dimostrare di essere l'autore del messaggio. Per farlo deve aggiungere al messaggio qualcosa che solo lui può conoscere. Si ricorre dunque al meccanismo della firma. Anche in

questo caso i calcoli per dimostrare la propria identità devono essere semplici, mentre devono essere computazionalmente impossibili quelli necessari alla creazione di un messaggio apparentemente autentico.

Tornando al meccanismo usato, diciamo che, come possiamo vedere in figura 3.3, la sorgente firma la checksum del messaggio e la spedisce. Se la destinazione riesce a decifrare la checksum e questa risulta uguale a quella che calcola dal messaggio, allora può esser sicuro dell'identità della sorgente.

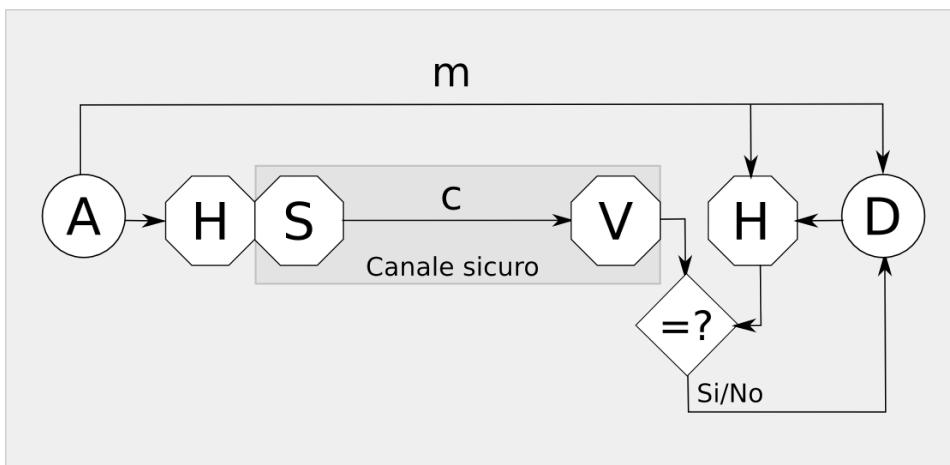


Figure 3.3: Autenticazione del mittente - 1

Un altro metodo consiste invece nel fare l'hash del messaggio concatenato col segreto. Lo vediamo in figura 3.4. In questo schema *A* e *B*, dopo essersi preventivamente accordati su un segreto *s*, lo usano per concatenarlo al messaggio e calcolare l'hash. Questo schema, al contrario del precedente è **ripudiabile**.

Un intruso non può costruire un messaggio che sembri scritto da un'altra persona se non conosce il suo parametro segreto. I calcoli per poter svolgere quest'attività illecita devono essere anche in questo caso molto difficili. L'autenticazione ottenuta secondo questo procedimento è importante perché fa sì che il messaggio sia **non ripudiabile**. Un meccanismo di autenticazione che invece risulta ripudiabile è quello in cui si utilizza un segreto condiviso da sorgente e destinazione. Affinchè non sia ripudiabile, un messaggio dev'essere firmato con qualcosa che solo l'autore può conoscere.

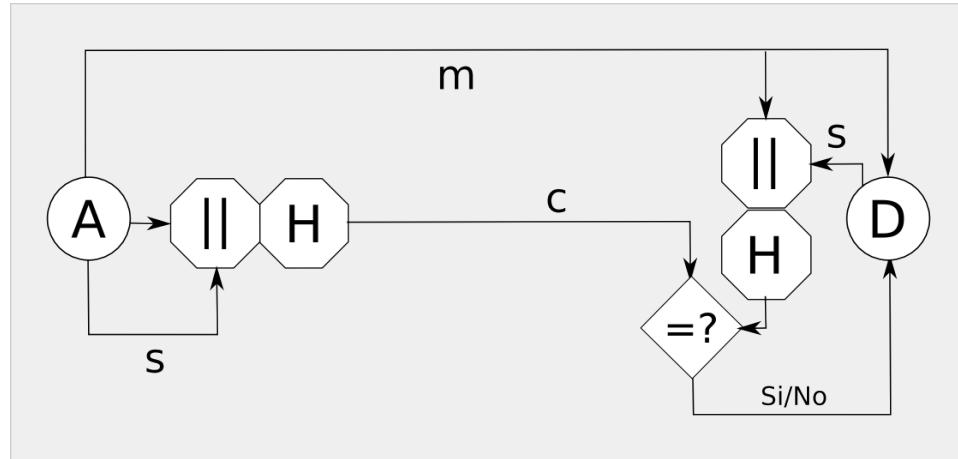


Figure 3.4: Autenticazione del mittente - 2

3.5 Identificazione

L'identificazione, al contrario dell'autenticazione, è utilizzata per identificare un utente a real time. Gli strumenti di identificazione che conosciamo sono diversi e si basano principalmente su uno o più di tre concetti:

- **conoscenza**: gli strumenti di identificazione basati sulla conoscenza, richiedono che l'utente che vuole autenticarsi conosca un segreto (ad esempio una password o un PIN).
- **possesso**: gli strumenti di identificazione tramite possesso verificano che l'utente sia in possesso di un particolare oggetto che può essere una smart card, o ancora una scheda a banda magnetica o un token.
- **conformità**: l'identificazione per conformità si basa solitamente su dati biometrici, ad esempio impronta digitale o dell'iride.

Un protocollo di identificazione prevede sempre come primo passo la **registrazione**, che è la fase in cui quelli che diventeranno rispettivamente **identificando** e **verificatore** si accordano sul segreto da verificare. Fatto ciò, il vero e proprio meccanismo di identificazione prevede tre fasi:

1. **dichiarazione**: l'identificando dichiara di voler essere identificato
2. **interrogazione**: il verificatore interroga l'identificando

3. **dimostrazione:** l'identificando comunica il suo segreto ed il verificatore lo confronta con quello atteso.

Le minacce ad un sistema di identificazione sono date dalla possibilità di dedurre il segreto, di replicarlo o ancora di rubare l'oggetto del segreto.

L'identificazione è univoca nel tempo. L'utente si identifica in un dato momento, dopo di che non permane lo stato. Al contrario l'autenticazione ha valore anche nel tempo.

3.6 I calcoli impossibili...

Abbiamo citato più volte la regola secondo la quale le operazioni lecite devono poter essere compiute con calcoli semplici, mentre le operazioni illecite devono essere possibili solo con calcoli difficili. Parlavamo della necessità di rendere impossibile ad un intrusore la compromissione dell'integrità o della riservatezza o fingere un'autenticazione. Impossibile è una parola grossa... Utilizzeremo la parola **difficile** (che spieghiamo a breve). Per ognuno degli scopi avremmo bisogno di funzioni unidirezionali.

Una funzione **f** è detta **unidirezionale** o **one-way function** se:

- è invertibile;
- facile da calcolare;
- per quasi tutti gli x appartenenti al dominio di **f** è **difficile** risolvere $y = f(x)$.

Un esempio è dato dall'elenco telefonico. Trovare il numero di telefono di una persona ha complessità $O(n)$, mentre trovare il nome di una persona conoscendone il numero ha complessità $O(2^n)$.

In teoria non esistono funzioni che hanno caratteristiche di unidirezionalità, ma nella pratica ne esistono molte che possono approssimare tale comportamento. Le funzioni **pseudo-unidirezionali** o **trapdoor one-way** appaiono come unidirezionali a chiunque non sia in possesso di una particolare informazione, un segreto.

Ok, ma sulla base di cosa definiamo un calcolo *difficile*? I problemi vengono classificati come **facili** se esistono algoritmi polinomiali in grado di risolverli su macchine di Turing deterministiche. I problemi sono ritenuti **difficili** se non sono stati finora individuati algoritmi che li risolvono in tempo polinomiale. Per chiarire meglio quanto appena detto introduciamo due grandezze:

- **Tempo di esecuzione di un algoritmo:** è il numero di operazioni N che occorre eseguire per terminarlo quando il dato di ingresso è rappresentato da una stringa di n bit. $N=f(n)$. In generale a parità di n si hanno diversi valori di N .
- **Tempo di esecuzione nel caso peggiore:** è il numero massimo di operazioni (N_{max}) che occorre eseguire per qualsiasi dato d'ingresso di n bit.

Sulla base di queste grandezze, possiamo dire che si studia l'andamento asintotico del tempo di esecuzione al crescere senza limiti di n e lo si definisce **Ordine di grandezza del tempo di esecuzione**: $T = O(g(n))$, dove $g(n)$ è una funzione tale che $0 \leq f(n) \leq c \cdot g(n)$ per $n \geq n_0$ e c costante. Se è nota l'espressione di $f(n)$ si prende come $g(n)$ il termine di $f(n)$ che cresce più rapidamente con n .

Definito $g(n)$ e definito l'origine di grandezza del tempo di esecuzione, possiamo dire che un algoritmo con tempo esponenziale ha $T = (\exp(n))$, mentre un algoritmo polinomiale ha $T = O(n^t)$ con t esponente più grande in $g(n)$.

Esistono anche algoritmi **sub-esponenziali**, che hanno come tempo di esecuzione $T = O(\exp((n)^\alpha(\ln(n)^{1-\alpha})))$ con $0 < \alpha < 1$. Anche questi sono considerati difficili.

Detto ciò possiamo riprendere il concetto più volte citato in precedenza e specificarlo meglio: ogni algoritmo che consente di **difendere** una proprietà critica dell'informazione deve avere tempo polinomiale, ogni algoritmo che consente di **violare** una proprietà critica deve avere tempo esponenziale. Utilizzeremo dunque funzioni pseudo-unidirezionali la cui risoluzione è polinomiale e la cui inversione è esponenziale o semi-esponenziale.

Possiamo ora calcolare il numero minimo di bit n necessari a garantire sicurezza. Prendendo come riferimento un calcolatore con velocità di un **MIPS** (*Million Instructions Per Second*), cioè 10^6 operazioni al secondo. L'unità di misura del tempo di un algoritmo di attacco si misura in **anni MIPS**. Un anno MIPS consiste in $10^6 \cdot 60 \cdot 60 \cdot 24 \cdot 365 = 31.536 \cdot 10^{12}$ operazioni. Dal grafico in figura 3.5 possiamo notare che al crescere dei bit n usati per il segreto, cresce il tempo necessario all'algoritmo di attacco. Nel 1998 la soglia di sicurezza era di 85 bit che corrispondeva a 10^{12} anni Mips. Oggi chiaramente il limite si è alzato per via della legge di Moore (secondo la quale la potenza di calcolo dei computer raddoppia ogni 18 mesi circa) e grazie al calcolo parallelo.

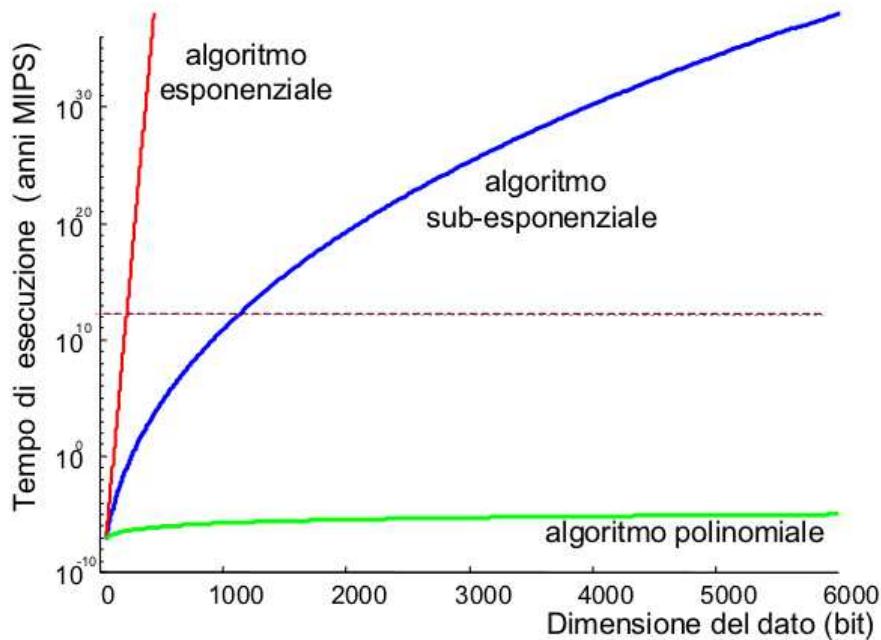


Figure 3.5: Il livello di sicurezza, complessità a confronto

La complessità di un algoritmo di attacco può sempre essere misurata facendo riferimento all'**algoritmo di ricerca esaurente** (*forza bruta*) che risolve tutti i problemi.

3.7 Trasformazioni segrete e chiavi

Abbiamo parlato di segretezza, ma non abbiamo specificato di cosa. Cos'è segreto? La macchina che viene utilizzata per produrre un'informazione sicura? Oppure l'algoritmo che viene utilizzato? O ancora il parametro passato all'algoritmo?

Riflettiamo un po'. Utilizzare delle macchine particolari per permettere la comunicazione sicura fra una sorgente ed una destinazione è una via non consigliabile per vari motivi. Il primo è che bisogna fidarsi del funzionamento della macchina, bisogna sapere come funziona e se davvero è in grado di garantire la sicurezza. Il secondo è che nel momento in cui viene scoperto il funzionamento è necessario riprogettare una nuova macchina e ciò porta a nuovi costi e perdite di tempo. Inoltre per comunicare con tante persone sarebbero forse necessarie tante macchine diverse. Inoltre esisterebbe

almeno qualcuno sulla terra in grado di compromettere la sicurezza della macchina: il creatore stesso. Per gli algoritmi i problemi son gli stessi. Se fossero segreti sarebbe necessario qualcuno in grado di certificare il loro funzionamento e sarebbe necessario trovare qualcuno che una volta che è stato scoperto l'algoritmo riesca a trovare un'alternativa presto. La cosa migliore è dunque demandare la responsabilità del segreto all'utente utilizzando un algoritmo pubblico ma che funzioni con un parametro privato, noto solo all'utente. In questo modo non c'è bisogno di macchine particolari, la valutazione dell'algoritmo è pubblica e il suo funzionamento può essere certificato facilmente, si abbattono i costi ed il funzionamento del meccanismo è molto più semplice e si eliminano i problemi indicati in precedenza.

Sulla base di quanto appena affermato, possiamo dire che i principali algoritmi che utilizzeremo per garantire la sicurezza sono algoritmi pubblici che funzionano con parametro segreto. Esistono due grandi classi di algoritmi, il primo è quello a **chiavi simmetriche**, il secondo quello a **chiavi asimmetriche**.

Iniziamo a vedere come si utilizzano le chiavi, vedendo il caso della riservatezza. Nella figura 3.6 possiamo notare che il mittente A utilizza una chiave k_s per cifrare il messaggio. Il destinatario B utilizza invece una chiave k_d per decifrarlo. Vediamo come devono essere le due chiavi:

- *algoritmo a chiavi simmetriche*: le due chiavi k_s e k_d sono uguali, quindi precedentemente mittente e destinatario si devono accordare sul segreto.
- *algoritmo a chiavi asimmetriche*: le due chiavi sono distinte. Ogni soggetto ha una chiave pubblica ed una privata. Volendo garantire la riservatezza, il mittente cifra il messaggio con la chiave pubblica del destinatario che userà la propria chiave privata per decifrarlo. Conoscendo la chiave privata dev'essere possibile calcolare facilmente la chiave pubblica. Conoscendo la chiave pubblica deve, chiaramente, essere difficile (impossibile) trovare la chiave privata.

Vediamo invece cosa è necessario fare per i due tipi di crittografia, per ottenere invece l'autenticazione.

- *algoritmo a chiavi simmetriche*: anche in questo caso le due chiavi sono identiche, quindi il mittente A esibisce a B il segreto per dimostrare di essere veramente chi afferma di essere.
- *algoritmo a chiavi asimmetriche*: in questo caso ancora una volta i due soggetti sono dotati ognuno di una coppia di chiavi pubblica e privata.

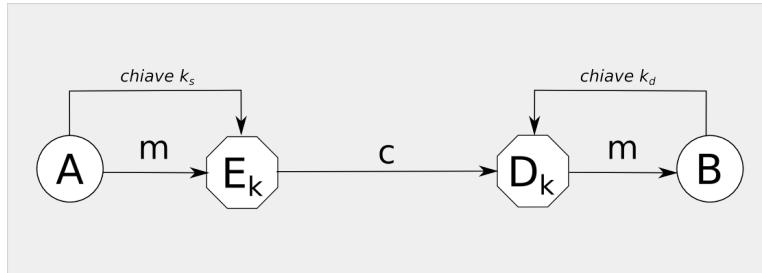


Figure 3.6: L'uso di chiavi per la riservatezza

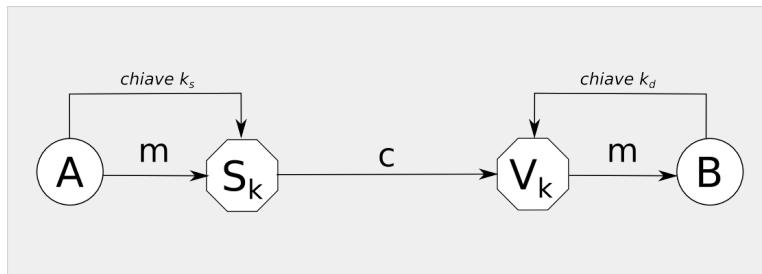


Figure 3.7: L'uso di chiavi per l'autenticazione

Questa volta, visto che A deve provare la sua identità, deve dimostrare di essere in possesso di conoscenze che altri non possono avere, dunque cifra il messaggio con la propria chiave privata. B utilizzerà la chiave pubblica di A per decifrare il messaggio e, se l'operazione andrà a buon fine, avrà in questo modo conferma dell'identità dell'interlocutore.

Vediamo quali proprietà devono essere garantite ad una chiave condivisa:

- **autenticità:** quando si concorda la chiave per via telematica è necessario essere certi che il corrispondente sia proprio voluto;
- **segretezza:** l'accordo sulla chiave deve avvenire su un canale a prova di intercettazione e il salvataggio della chiave deve avvenire su un supporto a prova di intrusione;
- **integrità:** la comunicazione della chiave e la sua successiva memorizzazione devono essere a prova di errore;
- **robustezza:** gli algoritmi di crittanalisi che individuano la chiave devono essere computazionalmente infattibili.

Diverso è il caso dell'algoritmo a chiavi asimmetriche: non c'è necessità di accordarsi sulle chiavi, in quanto ogni soggetto può generare autonomamente le proprie chiavi. Chiaramente la chiave privata dev'essere memorizzata, usata e custodita in maniera adeguata per evitare che intrusi possano leggerla o intercettarla. Dunque per una chiave privata le proprietà di sicurezza sono:

- **segretezza**: come detto, la chiave deve restare segreta;
- **integrità**: la memorizzazione dev'essere a prova di errore
- **robustezza**: non dev'essere possibile dalla chiave pubblica risalire alla chiave privata.

Mentre per una chiave pubblica, le due proprietà di sicurezza sono:

- **autenticità**: tutti devono poter avere la certezza che la chiave pubblica è proprio del soggetto desiderato;
- **integrità**: ancora una volta la memorizzazione dev'essere a prova di errore.

Per ottenere la chiave privata partendo da una chiave pubblica si deve usare una **one-way function**.

3.8 Crittanalisi

Abbiamo detto che la crittanalisi è la controparte della crittografia ed è la disciplina che studia come decifrare testi occultati. Nella crittanalisi si cerca di comprendere quali sono i segreti. Escludendo i calcoli, potremmo dire che i segreti in generale possono anche essere indovinati, intercettati o dedotti. Bisogna chiaramente rendere questo impossibile.

I segreti possono essere indovinati ad esempio facendo uso dell'algoritmo di ricerca esaurente (anche detto di forza bruta), che, provando tutte le combinazioni, prima o poi trova quella giusta. Un altro metodo consiste invece nell'utilizzare dei dizionari composti con le parole che più probabilmente sono state utilizzate per la password (nomi, cognomi, date ecc...). Dunque una contromisura è quella di comporre la propria password in maniera casuale. Inoltre, per difendersi anche dai tentativi di brute force è importante variare spesso la password. Gli algoritmi di forza bruta richiedono spesso molto tempo, dunque variare spesso la password fa sì che aumentino le probabilità che prima che l'algoritmo riesca a scoprire la password, questa sia già cambiata.

L’intercettazione della password può avvenire invece quando si utilizzano dei canali in chiaro per la trasmissione o quando si memorizzano i segreti in forma non cifrata. Dunque la password non solo dev’essere salvata in forma cifrata, ma deve anche essere scambiata con il processore che ne fa uso in maniera cifrata. Il processore dovrà poi cancellare dai registri ogni occorrenza del segreto una volta utilizzato.

L’ultima minaccia è rappresentata dalla deduzione. La deduzione può avvenire in vari modi. Utilizzando il segreto, siamo in ogni caso esposti a vari tipi di attacco:

- **con solo testo cifrato:** si studia il linguaggio in cui si pensa sia stato inviato il messaggio e si sfruttano i calcoli sulle probabilità di occorrenza.
- **con testo in chiaro noto:** se si hanno coppie di testo in chiaro e cifrato si può studiare il codice usato.
- **con testo in chiaro scelto:** se si ha la possibilità di avere la versione in chiaro di un determinato testo cifrato si può ricostruire il segreto.
- **con testo cifrato scelto:** se si ha la possibilità di avere testi in chiaro di testi cifrati scelti si può dedurre il segreto.

3.8.1 I tre livelli di gerarchia dei segreti

A proposito della possibilità di intercettare i segreti, vediamo comè possibile strutturare un file system cifrato. Lo faremo con quella che definiamo *gerarchia dei segreti*, una gerarchia composta da tre livelli.

L’utente sceglie e impara a memoria una password di cui viene memorizzato un hash. Primo livello.

Il terzo livello è quello dove vengono salvati i dati sensibili che l’utente vuole salvare in forma elettronica e sicura. I file posti al terzo livello sono archiviati facendo uso di una chiave scelta di volta in volta da un **RNG** (*Random Number Generator*). Il testo cifrato della chiave di cifratura è archiviato sullo stesso supporto.

Come si accede al terzo livello? Tramite una chiave memorizzata in un portachiavi e generata utilizzando un RNG ed un segreto (secondo livello).

Chiaramente questo metodo deve prevedere una qualche forma di recovery, altrimenti un eventuale smarrimento della passphrase o del portachiavi porterebbe alla perdita di tutti i dati riservati.

Capitolo 4

Dati sicuri - parte II

Esaminiamo i meccanismi, i servizi ed i dispositivi che creano le condizioni per impiegare correttamente i meccanismi con chiave.

4.1 Il generatore di numeri casuali

I generatori di numeri casuali sono chiamati in causa in tutti i meccanismi ed i servizi per la sicurezza. Alle stringhe in uscita dal generatore di numeri casuali è richiesto il rispetto di due proprietà:

- la **casualità**: ogni valore deve avere la stessa **probabilità di verificarsi** ed essere **statisticamente indipendente** da tutti gli altri;
- l'**imprevedibilità**: deve essere computazionalmente infattibile predire un valore da quelli che l'hanno preceduto, anche se si conosce perfettamente l'algoritmo o il dispositivo che ha generato la sequenza.

Per verificare la casualità sono stati definiti alcuni test statistici:

- **monobit**: valuta se il numero di *1* e *0* presenti all'interno della sequenza è approssimativamente lo stesso.
- **poker**: divide la sequenza in blocchi da M bit e valuta se il numero di volte che compare ciascuna delle 2^M configurazioni è approssimativamente lo stesso;
- **run**: considera le stringhe di bit consecutivi formate da tutti *1* (*block*) o da tutti *0* (*gap*) e valuta per varie lunghezze se il loro numero approssima il valore atteso per una vera sequenza casuale;

- **long run:** considera il più lungo run di 1 presente nella sequenza in esame e valuta se la sua lunghezza è compatibile con quella attesa per una sequenza casuale.

Altri metodi si basano sul calcolo di particolari funzioni della stringa, come l'**autocorrelazione** (controlla bit a bit di quanto variano la sequenza originale e quella traslata di una certa quantità), la **trasformata discreta di Fourier** (controlla se esistono pattern ripetitivi troppo vicini uno all'altro).

L'**imprevedibilità** viene verificata tramite il **next-bit test**: dati L bit della stringa casuale, non deve esistere alcun algoritmo di complessità polinomiale che permetta di predire il bit $L+1$ con probabilità significativamente maggiore di 0,5.

4.1.1 Il True Random Number Generator

Il vero generatore di numeri casuali si basa su **fenomeni fisici** che intrinsecamente includono un buon livello di casualità. Esempi di questi fenomeni fisici sono il tempo medio di emissione di particelle durante il decadimento radioattivo, il rumore termico ai morsetti di un apparato... In questi **generatori hardware** si effettua una digitalizzazione di un segnale analogico fornito dalla sorgente. A volte viene poi fatta una fase di post-processing (necessaria a volte per rendere equiprobabili 1 e 0 , *de-skewing*) della stringa e la memorizzazione del dato per poi fornirlo alle applicazioni. I **True Random Number Generator (TRNG)** hanno buone caratteristiche di casualità. In generale tali dispositivi possono anche essere implementati via software, magari estraendo rumore da componenti come l'hard disk a patto però di mantenere le caratteristiche di casualità ed imprevedibilità. Uno svantaggio dei *TRNG* sta nel fatto che a volte la frequenza di produzione dei dati casuali può essere troppo bassa rispetto al necessario.

4.1.2 Lo Pseudo Random Number Generator

Esistono anche generatori algoritmici, detti **pseudocasuali (PRNG)** dato che la sequenza in uscita prima o poi si ripete. Rispetto ad un TRNG, dunque abbiamo lo svantaggio di avere una certa periodicità, inoltre le sequenze possibili sono inferiori ed il valore di uscita risulta prevedibile. Gli pseudo random number generator seguono il modello di macchina a stati finiti visibile in figura 4.1. Il registro di stato emette il valore da mandare in uscita, il valore corrispondente allo stato presente F . Detto valore viene inviato anche per la determinazione di G (stato futuro).

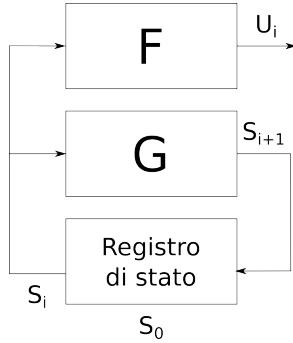


Figure 4.1: FSM di un PRNG

Uno degli algoritmi più usati per la generazione di numeri casuali è il **metodo di congruenza lineare**: l'algoritmo si basa sull'iterazione del calcolo $X_{i+1} = (a \cdot X_i + b) \bmod m$ dove m è un numero primo mentre a e b sono numeri interi. Se a e b sono stati scelti opportunamente, la casualità è buona. Non buona è l'imprevedibilità, in quanto bastano pochi X_i per individuare m , a e b , dunque ogni altro successivo X .

Oggi si ritiene conveniente l'uso di generatori algoritmici, perciò è stato necessario trovare nuovi algoritmi per conseguire **casualità** e **imprevedibilità**. Questi algoritmi possono fare uso di meccanismi propri della crittografia simmetrica o basarsi su uno dei problemi della crittografia asimmetrica. La **crittografia simmetrica** impiega il modello dell'automa a stati finiti che assume uno dopo l'altro tutti i suoi possibili stati (il cui numero dev'essere elevatissimo, anche oltre 10^{50}). La funzione di stato futuro e quella di uscita devono essere unidirezionali ed il seme dev'essere imprevedibile e tenuto segreto. In particolare questo sarebbe consigliabile farlo generare ad una vera sorgente di rumore. Questo modello è ad alta velocità e la sua sicurezza crittografica è stata verificata sperimentalmente. Nel caso della crittografia asimmetrica anche si mettono in gioco dei segreti, ma le trasformazioni richieste sono più complesse, dunque hanno performance più limitate, ma offrono in cambio una sicurezza crittografica teoricamente dimostrata.

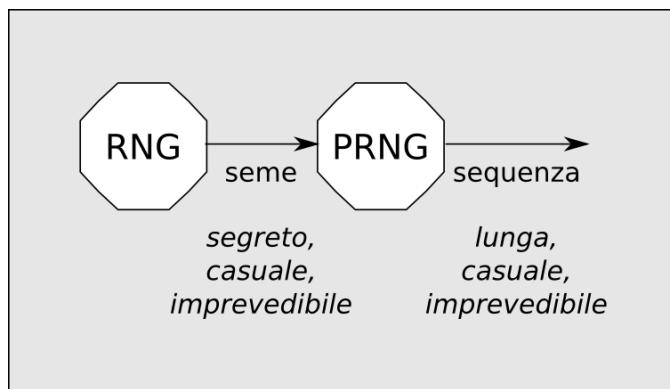


Figure 4.2: Un PRNG crittograficamente sicuro

4.2 La funzione Hash

Come abbiamo detto precedentemente, una funzione hash comprime una stringa di m caratteri in una di h di lunghezza fissa e relativamente piccola: $h = H(m)$

L'uscita della funzione hash è detta **riassunto** o **impronta** (*digest* o *fingerprint*).

Una funzione di hash che restituisce una stringa di h bit, non fa altro che dividere l'insieme delle possibili combinazioni in ingresso (2^h) in altrettanti insiemi, ognuno dei quali contiene le stringhe che producono un'uguale impronta. Come sappiamo infatti (e come possiamo immaginare) ridurre la dimensione del dominio da m ad h fa sì che alcune stringhe producano lo stesso hash, generando dunque delle **collisioni**. A questo proposito si distinguono le funzioni hash fra **semplici** e **critografiche** a seconda che l'individuazione di collisioni sia facile o difficile. Per proteggere l'**integrità** è necessario che l'individuazione di collisioni sia computazionalmente impossibile. Per proteggere invece la **riservatezza** dev'essere computazionalmente impossibile il calcolo dell'inversa. In ogni caso il calcolo di $H(x)$ dev'essere facile.

L'impossibilità di individuare una collisione è indicata come **robustezza debole** e definita da: "*per ogni x dev'essere difficile trovare un $y \neq x$ tale che $H(y)=H(x)$* ".

Esiste anche la **robustezza forte** ed è definita da: "*dev'essere computazionalmente infattibile trovare una qualsiasi coppia x,y tale che $H(y)$ sia uguale a $H(x)$* ".

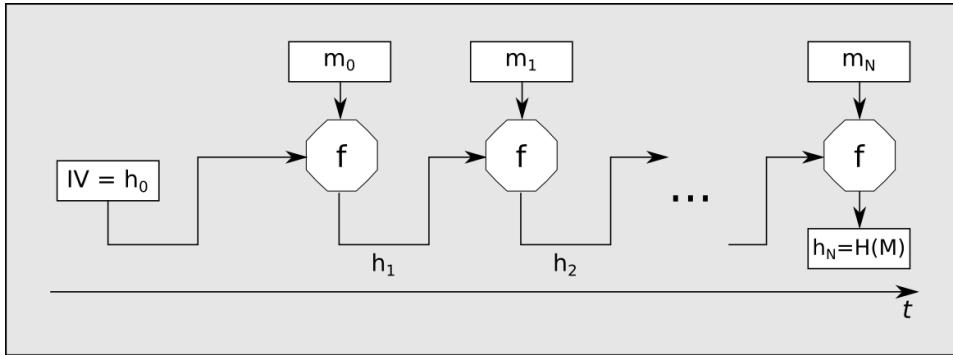


Figure 4.3: Schema di compressione iterata

La terza proprietà importante che devono avere le funzioni hash è la **non-invertibilità**.

Tutti gli algoritmi di hashing si basano sul principio della **compressione iterata**. Il messaggio M viene diviso in blocchi di lunghezza r . Il nucleo centrale è una funzione f che genera n bit in uscita a partire da due dati in ingresso, uno di r bit ed uno di n ($r > n$). Nell'ultimo blocco estratto dal messaggio viene inserito un *padding* di 0 se la dimensione non raggiunge r . Il risultato in uscita da ogni blocco f è detto h_i e va in ingresso al successivo. Il valore h_0 è il **vettore di inizializzazione**, mentre h_N è l'impronta di M . La funzione f deve essere resistente alle collisioni e tale resistenza si ottiene sottoponendo i dati in ingresso a diverse elaborazioni. Tutto è più chiaro con uno sguardo alla figura 4.3.

4.2.1 Attacco alle funzioni di hash

Un attacco a cui sono vulnerabili alcune funzioni di hash è il **length extension attack**: conoscendo $h(m)$ e $\text{len}(m)$, l'attaccante può individuare un opportuno m' e calcolare $h(m//m')$. In pratica osserviamo dallo schema in figura 4.3, che la chiave viene utilizzata soltanto nel primo blocco, quelli successivi operano con i valori in uscita dai blocchi precedenti, perciò si può pensare di ascoltare il canale, recuperare $h(m)$ e usarlo come ingresso per un altro blocco, in modo tale da riuscire a concatenare m' e ottenere un hash valido.

Un altro attacco può essere perpetrato tramite collisioni. Se siamo in grado di individuare una collisione, cioè un messaggio m' che produce lo stesso hash di m , allora possiamo sfruttare questa collisione per far autenticare m' anzichè m . Una contromisura consiste nel fare l'impronta dell'im-

pronta per ridurre le possibilità di individuare una collisione.

Dato un messaggio m , il calcolo di una collisione, cioè l'individuazione di un messaggio m' tale che gli hash siano uguali, è un problema di complessità esponenziale. La soglia di sicurezza attualmente supera gli 80 bit.

Potendo scegliere a caso anche il messaggio m , il calcolo di una collisione diventa più efficace. Ciò segue da un paradosso statistico secondo il quale volendo trovare una persona nata in una specifica data occorrono 2^{53} persone affinché la probabilità sia maggiore di 0,5, mentre volendo trovare due persone nate lo stesso giorno (un giorno a caso) ne occorrono solo 23. L'algoritmo che risolve il secondo problema richiede un numero di tentativi proporzionali a $2^{\frac{n}{2}}$, mentre quello per il primo 2^n .

Ma come si sfrutta l'individuazione di due messaggi a caso che producono lo stesso hash? Negli schemi di firma digitale che consente di garantire l'autenticità e l'integrità di un documento, si firma soltanto l'hash. Sfruttando il **birthday attack**, un malintenzionato può affermare di aver inviato un dato messaggio, mentre in realtà ne è stato inviato un altro (che però ha lo stesso hash). Facciamo notare però che il malintenzionato potrebbe anche essere l'identificatore... Vediamo ora come funziona l'attacco citato:

1. il malintenzionato genera 2^n messaggi, uno diverso dall'altro, ottenuti tramite modifiche al messaggio originale;
2. calcola e memorizza l'hash di tutte le versioni;
3. introduce nuove modifiche al secondo messaggio, calcola l'hash e verifica se è presente in memoria. In caso negativo ripete il passaggio. Per il paradosso del compleanno può aspettarsi di trovare una coincidenza dopo $2^{\frac{n}{2}}$ iterazioni.

Oggi il meccanismo di firma digitale con appendice (cioè autenticazione con hash del solo documento) ha oggi valore legale, pertanto è importante usare funzioni hash da 160 bit che abbiano resistenza forte alle collisioni.

4.2.2 Gli algoritmi di hash

Ad oggi gli algoritmi più noti sono **MD5**, **SHA-1**, **RIPEMD-160**. Mentre il primo opera con 128 bit, gli altri fanno uso di 160 bit. Tutti operano con blocchi da 512 bit. SHA-1 ha un limite nella dimensione massima del messaggio ($2^{64} - 1$ bit), mentre gli altri non hanno limiti.

Altri algoritmi sono Tiger, SHA-256, SHA-384, SHA-512...

4.3 Servizi di identificazione

Un protocollo di identificazione ha lo scopo di identificare un'entità A verso un'entità B . Un protocollo del genere deve però rispettare alcuni requisiti:

- se le entità in gioco sono fidate, B deve poter completare il protocollo di identificazione certo dell'identità di A ;
- B non può riutilizzare lo scambio di identificazione avuto con A per impersonare illegittimamente A ;
- la probabilità che un soggetto C riesca a eseguire e completare il protocollo spacciandosi per A deve essere prossima allo 0;
- tutti gli obiettivi citati finora devono rimanere validi se un numero elevato di identificazioni tra A e B sono state osservate e se C è stato precedentemente coinvolto in sessioni di identificazione con A o B .

Un protocollo di identificazione si articola secondo tre semplici passi (previa fase di registrazione in cui A e B si sono accordati circa la prova d'identità):

1. L'entità A che si vuole identificare dichiara la sua volontà a B
2. B interroga A
3. A fornisce una prova della sua identità a B

L'identificazione si divide in **passiva** (o *debole*) e **attiva** (o *forte*). L'identificazione passiva si basa solitamente sull'inserimento di una password ed è solo unilaterale, non può essere usata per identificazione reciproca. Quella forte invece può essere usata in entrambe le modalità.

4.3.1 L'identificazione passiva

Un meccanismo di identificazione passiva, come accennato precedentemente, si basa sull'inserimento di una password. Un soggetto A che vuole essere identificato presso una macchina M invierà un messaggio $A//psw A$, cioè comunicherà la sua identità e fornirà la password (che deve conservare gelosamente).

Un intruso I può provare a **indovinare**, **intercettare** o **leggere** la password. Alcuni meccanismi di protezione devono dunque essere adottati. Un esempio di questi è dato dal non stampare a video la password durante

l'inserimento per evitare sguardi indiscreti... Nell'identificazione si presuppone che il meccanismo di identificazione non sia dotato di sue risorse di elaborazione, dunque è da scartare la soluzione di cifrare la password prima dell'invio (e in ogni caso non proteggerebbe da attacchi di replica).

Una difesa contro la possibilità di indovinare la password consiste nello scegliere password complicate o nel fare in modo che sia fornita all'utente una password sicura generata automaticamente. Inoltre l'utente deve evitare di trascrivere la password dove questa potrebbe essere rubata.

Ora che abbiamo visto le vulnerabilità dell'utente, passiamo ad analizzare le vulnerabilità del sistema. Se l'intruso riesce ad accedere al file delle password i problemi sono seri. Ad esempio potrebbe aggiungere in coda il suo ID, l'hash della sua password e comunicare le autorizzazioni di cui ha bisogno. Occorre dunque memorizzare in una forma cifrata comprensibile solo al proprietario del file le informazioni e solo il proprietario deve poter accedere al file in lettura e scrittura. Notiamo che salviamo l'hash e non la password, per proteggere ulteriormente il segreto. La non invertibilità dell'hash fa sì che anche l'amministratore del sistema non possa risalire alla password, perciò solo l'utente conosce la sua password. Se le password sono facili si può comunque fare un **attacco con dizionario** che consiste nel calcolare l'hash delle voci contenute in un dizionario ed effettuare delle varianti, calcolare nuovamente gli hash e provare fino a che non trova una password che dia lo stesso hash. Per rendere l'attacco più complicato, il sistema Unix usa il **salt**, cioè un numero casuale che estrae per ogni nuova password. La password viene concatenata con il salt e poi si calcola l'hash. In questo modo lo spazio dei valori possibili è molto più ampio. Aggiungere p bit di salt genera 2^p differenti termini di paragone di una stessa password e chi vuole condurre un attacco è obbligato a provarli tutti.

Un problema intrinseco dell'identificazione passiva è quello per cui A deve essere certo a priori dell'identità di B ! Per risolvere si potrebbe ricorrere ad un meccanismo di identificazione reciproca, ma ciò non è possibile nel caso di identificazione passiva.

4.3.2 L'identificazione attiva

L'identificazione passiva è soggetta a problemi quali l'attacco di replica. La vera contromisura a quest'attacco consiste nel cambiare di continuo la prova d'identità. Si parla dunque di **identificazione attiva** (o forte dato che restiste agli attacchi di replica e anche perché in questo modo si può realizzare la mutua identificazione).

L'identificazione attiva può essere realizzata con tre diversi principi:

- one-time password
- sfida/risposta
- zero knowledge

Tutti devono soddisfare la regola: il calcolo della prova di identità da fornire di volta in volta deve essere facile per chi conosce un’informazione segreta, difficile per chi dispone solo delle prove inviate in precedenza.

4.3.3 One-time password

Il metodo di cifratura one-time può funzionare o tramite funzioni unidirezionali o tramite un cifrario con chiavi di sessione.

Il metodo della password usata una sola volta è molto semplice. Supponendo che A voglia identificarsi presso la macchina M , dovrà concordare con esso una serie di password da utilizzare. In fase di registrazione A sceglie un numero casuale X_A ed impiega una funzione non invertibile F per calcolare gli n valori $X_A, F(X_A), F^2(X_A), \dots, F^{n-1}(X_A), F^n(X_A)$ che memorizzerà in ordine inverso. Su M viene memorizzato soltanto $F^n(X_A)$.

Come funziona l’identificazione? A spedisce a M la chiave $F^{n-1}(X_A)$ ed M che conosce la trasformazione F la applica per ottenere $F^n(X_A)$. Se il risultato corrisponde al valore atteso allora A è identificato. Terminata l’identificazione, M memorizzerà al posto di $F^n(X_A)$ il valore $F^{n-1}(X_A)$ ed A , per identificarsi nuovamente, dovrà inviare $F^{n-2}(X_A)$, M calcolerà $F^{n-1}(X_A)$ e via dicendo fino all’esaurimento delle n password.

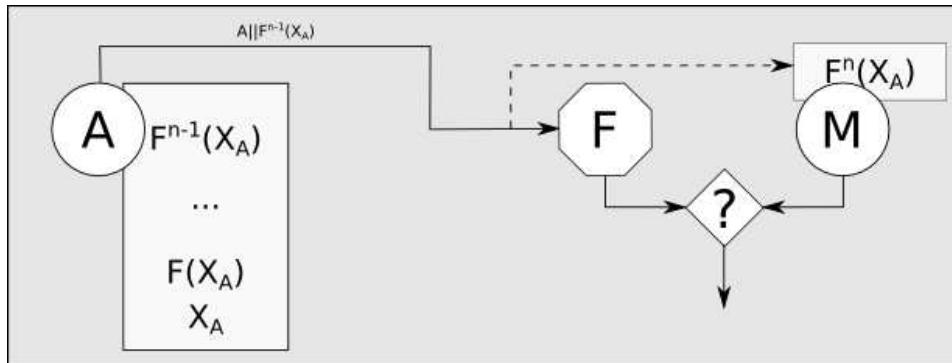


Figure 4.4: la cifratura one-time

L'invio di password sempre diverse può avvenire anche tramite un cifrario: la password memorizzata rimane sempre la stessa, ma i due corrispondenti, all'inizio di ogni sessione modificano in segreto la chiave.

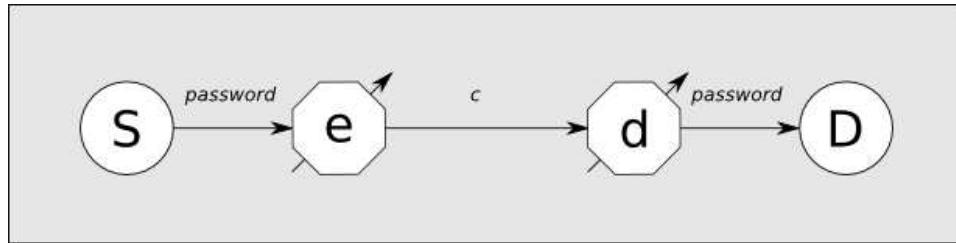


Figure 4.5: Cifratura One-Time

4.3.4 Sfida e risposta

Oggi è il metodo di identificazione attiva più utilizzato. Per eseguirlo, i due corrispondenti possono usare o funzioni **hash sicure** o **cifrari** (simmetrici o asimmetrici) o ancora uno schema di firma digitale.

Caso con funzione hash: A e B si accordano preventivamente su una funzione hash da utilizzare e su un segreto s . Quando A vuole essere identificato informa B che gli invia un **nonce** (un numero valido una sola volta) R_B . A risponde inviando a B il messaggio $H(R_B||s)$. B verifica il valore atteso e decide.

Se si vuole identificazione mutua, il protocollo varia leggermente:

- B invia ad A un nonce R_B ;
- A invia un messaggio con il nonce di B cifrato ed un suo nonce: $R_A||H(R_B||s)$.
- B cifra un messaggio contenente il nonce di A ed il segreto: $H(R_A||s)$.

In realtà questo metodo è vulnerabile all'attacco di **reflection** che sfrutta il problema del **gran maestro di scacchi**. Un giocatore C vuole apparire come un gran giocatore contro un gran giocatore A pur non sapendo giocare. Come farà? Inizia una partita con un altro grande giocatore, B , dal quale copia le mosse per poi riprodurle con A . Modifichiamo dunque il meccanismo in modo tale che C non possa introdursi fra A e B . La soluzione sta nell'introdurre l'identificativo del destinatario nella risposta.

- B invia ad A un nonce R_B ;

- A invia un messaggio a B : $R_A||H(R_A||R_B||B||s)$
- B cifra un messaggio contenente i due nonce, l'identità di A ed il segreto: $H(R_A||R_B||A||s)$.

Si possono usare inoltre dei timestamp per verificare il tempo ed evitare che l'intruso ne abbia per attuare l'attacco. Anche i numeri di sequenza sono un utile accorgimento.

Caso con cifrario: nel caso in cui si voglia usare un cifrario, B invia un nonce cifrato ad A che lo decifra e lo reinvia a B . In questo modo B potrà verificare l'identità.

Caso con firma digitale: B invia il nonce ad A che lo firma con la sua chiave privata e lo reinvia. Se B riesce a decifrarlo tramite la chiave pubblica di A allora l'identificazione riesce.

4.3.5 Zero knowledge

Alla base dei protocolli **zero-knowledge** c'è il principio di dare solo una testimonianza di saper risolvere facilmente un problema da tutti ritenuto difficile.

1. L'identificando fornisce una **testimonianza** di ciò che sa fare;
2. il verificatore lancia una sfida che possa convincerlo sulla veridicità della testimonianza;
3. l'identificando invia la risposta.

In realtà tutto è più comprensibile con un esempio, l'esempio della caverna di Ali Babà: una caverna è formata da due tunnel separati da una porta che può essere usata solo usando una frase segreta. Ali Babà vuole convincere il califfo Omar di essere a conoscenza del segreto senza però rivelarglielo. A tal fine Ali Babà dice ad Omar di restare nel tunnel nel punto A ed entra in uno dei due. A questo punto Omar avanza fino al punto B e chiede ad Ali di uscire o dal tunnel di destra o di sinistra. Ali ci riesce aprendo se necessario la porta con la frase segreta. Un impostore ha solo il 50% di probabilità di trovarsi già nel tunnel giusto. Ovviamente la prova andrà iterata un po' di volte. Dopo n ripetizioni della prova, la probabilità che Ali sia un impostore è 2^{-n} .

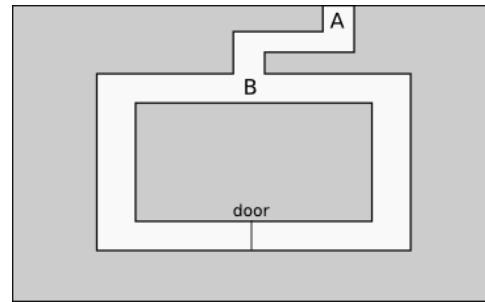


Figure 4.6: la caverna di Alì Babà

Capitolo 5

Cenni di crittologia classica

La riservatezza di un'informazione può essere conseguita tramite la **steganografia** o la **crittografia**. Le due tecniche sono differenti: nella prima si nasconde l'informazione (ad esempio si cela un testo in un'immagine), nella seconda la si rende incomprensibile.

La crittografia, che sarà il nostro argomento di studi, prevede l'utilizzo di **codici** o di **cifrari**. Oggi vengono utilizzati soltanto i cifrari, in quanto più sicuri.

I **codici** associano ad ogni simbolo dell'alfabeto di partenza, un simbolo di un altro alfabeto. Ad esempio in tabella possiamo notare un alfabeto di partenza (il nostro) e un alfabeto di arrivo ottenuto senza alcun criterio particolare:

Alfabeto originale:	a b c d e f g h ...
Alfabeto di uscita:	p y f g c r l a ...

Se A e B vogliono comunicare, devono preventivamente scambiarsi l'alfabeto da utilizzare.

Differenti invece sono i **cifrari** che sono composti anch'essi da trasformazioni di simboli in ingresso con differenti simboli in uscita, ma tramite sostituzioni guidate da **chiavi**. Ad esempio è possibile realizzare una trasformazione dicendo che ad ogni simbolo i dell'alfabeto viene sostituito il simbolo in posizione $i+3$ (esempio noto come *cifrario di Cesare*). In questo caso non vi è la necessità di inviare l'alfabeto di uscita, ma basta scambiare soltanto la chiave, ovvero il numero 3. In generale comunque, per garantire un livello

di sicurezza adeguato, vengono effettuate oltre alle **sostituzioni**, anche delle **trasposizioni**.

5.1 Cifrari classici

5.1.1 La sostituzione monoalfabetica

Si definisce **sostituzione monoalfabetica** il metodo di crittografia nel quale si effettua una sola operazione di sostituzione fra i simboli di un alfabeto di partenza e quelli di un alfabeto di arrivo ottenuto tramite una chiave. Ad esempio nel cifrario di Cesare la chiave è 3, nel cifrario ATBASH si sostituisce alla prima lettera l'ultima, alla seconda la penultima e così via. Questi cifrari sono particolarmente deboli. Dato un alfabeto di n simboli, il numero di trasformazioni possibili sono $n!$.

Un metodo di attacco che può essere perpetrato in caso di messaggi di partenza in linguaggio naturale è l'**attacco statistico**. Ogni linguaggio usa i caratteri dell'alfabeto con frequenze diverse e largamente dipendenti dalla semantica delle parole usate. Sfruttando la conoscenza delle statistiche con cui compaiono i simboli in una lingua è possibile più facilmente capire il testo inviato. Ciò è possibile perché in una lingua vi è una insita ridondanza e la probabilità di occorrenza di stringhe corte è indipendente dal testo. In figura 5.1.1 possiamo vedere come funziona questo tipo di attacco.

- 1) testo inglese cifrato con una sostituzione monoalfabetica
UNUFT OST, SII QNUF RBU GQFIO, HQDWXRUF KURGQFVL SKO BQLR ...
- 2) analisi frequenziale dei caratteri del testo cifrato:
U 15,3%, **R** 9,8% , **S** 7,8%
UF, FU e RB 3,3%,
RBU 3,5%, **USV** 2%.
S spesso da sola e 13% come prima lettera di una parola
- 3) ipotesi: **U** \leftrightarrow **E**, **R** \leftrightarrow **T**; conferma: **RBU** \leftrightarrow **THE**; conseguenza: **B** \leftrightarrow **H**.
- 4) sostituzioni:
ENEFT OST, SII QNEF THE GQFIO, HQDWXTEF KETGQFVL SKO HQLT
- 5) nuove ipotesi: **S** \leftrightarrow **A** , **UF** \leftrightarrow **ER**, **FU** \leftrightarrow **RE** ; conseguenza: **F** \leftrightarrow **R**.
- 6) sostituzioni:
ENERT OAT, AII QNER THE GQRIO, HQDWXTER KETGQRVL AKO HQLT..
N.B. 18 caratteri su un totale di 46, cioè circa il 40%.
- 7) Statistiche non ancora prese in considerazione e significati probabili :
EVERY DAY, ALL OVER THE WORLD, COMPUTER NETWORKS AND HOSTS...

Figure 5.1: Rottura di un cifrario monoalfabetico

Il più grosso limite dei cifrari monoalfabetici è quello di trasferire le proprietà statistiche di occorrenza di ogni carattere del testo in chiaro al carattere che lo sostituisce.

Per cercare di risolvere i problemi appena presentati, si applicano alcuni espedienti:

- si eliminano dal testo in chiaro le spaziature e i segni di interpunkzione;
- si introducono nel testo in chiaro dei caratteri non significativi (le cosiddette *nulle*);
- si impiega nel testo cifrato un alfabeto più grande di quello usato nel testo in chiaro;
- si sostituiscono due o tre caratteri alla volta (cifrari poligrafici).

La sostituzione monoalfabetica viene oggi applicata a blocchi di minimo 8 caratteri (a cui corrispondono, considerando una codifica ASCII, 64 bit). Ciò è sufficiente a prevenire l'attacco statistico in quanto è computazionalmente impossibile raccogliere e memorizzare 2^{64} dati statistici.

5.1.2 Sostituzione di digrammi: il cifrario Playfair

Il **cifrario Playfair** si basa sull'uso di una matrice 5x5 contenente una parola chiave. Nella tabella viene inserita la parola chiave eliminando le lettere duplicate e si riempie il resto della tabella con le lettere restanti dell'alfabeto. Dato che le lettere nell'alfabeto inglese sono 26, solitamente si considerano *I* e *J* come una lettera sola. È un meccanismo di sola sostituzione.

Per cifrare un messaggio si divide il messaggio in digrammi (gruppi di due lettere). Se le lettere sono dispari si aggiunge il padding, ad esempio con una X. Le lettere di un digrafo individuano un rettangolo nella tabella che ha per vertici opposti le due lettere. A quel punto si sostituisce al digrafo individuato, un altro digrafo costituito dalle altre due lettere nel quadrato. Se invece le lettere del digrafo di partenza sono sulla stessa riga, si codificano con le lettere alla propria destra (considerando ciclica la tabella), se sono sulla stessa colonna si prendono le lettere subito sotto (anche in questo caso considerando ciclica la tabella). Se le lettere nel digrafo di partenza sono uguali, si aggiunge una X al posto della seconda lettera e si cifra quel blocco.

Per decifrare si usa lo stesso algoritmo all'inverso eliminando ogni X che è necessaria nel messaggio finale.

Esempio: vediamo il caso in cui la chiave sia "*esempio*" e la frase da cifrare sia "*viva la crittografia*". La tabella appare come segue:

e	s	m	p	i
o	a	b	c	d
f	g	h	k	l
n	q	r	t	u
v	w	x	y	z

Dividiamo la frase da cifrare in *digrammi*: *vi va la cr it to gr af ia*.
La frase crittografata è: *dp go gd hk kd ey hq og sd*.

5.1.3 La trasposizione di colonne

Vediamo ora un altro metodo interessante. Data una frase da cifrare, è possibile inserirla in una tabella $P \times Q$, riempiendo con eventuali X di padding le celle vuote. A questo punto si stabilisce un codice numerico composto dai numeri da 1 a Q , presi una e una sola volta. Il testo cifrato sarà ottenuto prelevando i simboli da ogni colonna, prendendo le colonne nell'ordine indicato dalla chiave. Si tratta di un meccanismo di sola trasposizione.

Esempio: la frase da cifrare è ancora *viva la crittografia*. Sceliamo $P=5$, $Q=4$. Come chiave prendiamo *25341*. La tabella è:

1	2	3	4	5
v	i	v	a	l
a	c	r	i	t
t	o	g	r	a
f	i	a	X	X

Da cui segue che la frase cifrata è: *icoi ltax vrga airx vatf*.

Anche qui l'analisi statistica può aiutare, ma l'unica informazione utile che possiamo avere consiste nell'osservare quali digrammi non sono naturali, ma derivano da scrittura in colonne del testo in chiaro. In questo modo potremmo riuscire a ricostruire il testo di partenza.

5.1.4 Sostituzione polialfabetica o cifrario di Vigenère

L'obiettivo della sostituzione polialfabetica è quello di rendere equiprobabile l'occorrenza di ogni simbolo nel testo di modo tale che l'analisi statistica non possa fornire alcun aiuto.

In questo metodo si crea una tabella in cui nella prima riga si scrive in ordine ogni simbolo dell’alfabeto. Nella seconda si ripete il tutto traslando a sinistra di una posizione, nella terza lo stesso e via dicendo.

A	B	C	D	E	F	G	H	I	J	K	L	...
B	C	D	E	F	G	H	I	J	K	L	M	...
C	D	E	F	G	H	I	J	K	L	M	N	...
D	E	F	G	H	I	J	K	L	M	N	O	...
...

La frase cifrata (supponendo ancora una volta che la frase originale sia *viva la crittografia* e supponendo che la chiave sia *chiave*) si ottiene con questo semplice procedimento: per cifrare si sceglie la riga della tabella che inizia con il carattere di chiave e si cerca la colonna che ha come intestazione il carattere da cifrare e si fa l’incrocio.

In questo caso la frase cifrata è: *argah ebstrs ibifge*.

Il punto debole della sostituzione polialfabetica sta nel fatto che se la chiave è lunga L caratteri, ogni L simboli si userà la stessa sostituzione alfabetica.

In questo metodo di cifratura, l’attacco statistico non è inutile come si è pensato per lungo tempo. La debolezza di questo metodo sta nel fatto che a tutti gli effetti è un insieme di n cifrari di Cesare (dove n è la lunghezza della chiave). Se il crittoanalista riesce a determinare la lunghezza della chiave, diventa molto semplice decifrare il testo. Il metodo **Kasiski** consiste nell’individuare sequenze ripetute. Con buona probabilità, il massimo comune divisore fra le distanze di queste sequenze (o un multiplo) corrisponderà alla lunghezza della chiave. **Babbage** ha formulato ma mai pubblicato un algoritmo analogo per risolvere il problema. Anche **Friedman** ha formulato un algoritmo per la rottura di questo cifrario, ma basato su concetti differenti, come l’indice di coincidenza.

Gli accorgimenti per rendere più sicuro questo metodo consistono nello scegliere chiavi lunghe e casuali, nel non archiviare insieme testi cifrati e decifrazioni.

5.2 Il cifrario Vernam One Time Pad

Il cifrario di Vernam è ancora una volta un metodo di sostituzione polialfabetica. In questo caso però si trasformano i simboli della chiave e della frase da cifrare usando la codifica Baudot (5 bit). Nella tabella dobbiamo come al solito incrociare una riga ed una colonna per ogni simbolo e trovare il gruppo di bit corrispondente. Questo si può ottenere anche senza tabella ma con uno XOR fra le due cinquine di bit. In questo algoritmo per aumentare la sicurezza si fa uso di una chiave lunga quanto tutto il testo, inoltre, a seguito di analisi di Mouborgne, si fa durare la chiave per un solo giro e la si sceglie in maniera casuale. La scelta di una chiave casuale rende invulnerabile ad attacchi statistici.

Il cifrario di Vernam-Mouborgne è stato rinominato **one-time pad**. Non può essere violato con attacchi passivi, per questo viene definito il **cifrario perfetto**. È tuttavia vulnerabile ad attacchi attivi in quanto se sappiamo la firma inviata e intercettiamo la firma codificata, possiamo tramite XOR ricavare la chiave da quei bit e utilizzarla per inserire un nuovo nominativo della stessa lunghezza.

Emerge una perplessità... Come trasmettiamo la chiave se è lunga quanto tutto il messaggio? Serve un canale sicuro e disponibile per l'intera lunghezza del messaggio, lo stesso che potremmo in alternativa usare per trasmettere direttamente il messaggio.

5.3 Definizioni di sicurezza

5.3.1 Sicurezza dei cifrari

Un cifrario è detto **perfetto** o **assolutamente sicuro** se, dopo aver intercettato un certo testo cifrato C , l'incertezza a posteriori sul testo in chiaro M corrisponde all'incertezza che si aveva a priori, cioè prima dell'intercettazione. Il cifrario è detto **sicuro** se dato un testo cifrato C , trovare M tale che $E_k(M) = C$ è impossibile per chi non conosce k . Un cifrario è detto **computazionalmente sicuro** se calcolare M da C è possibile, ma richiede una potenza di elaborazione superiore a quella a disposizione dell'attaccante.

5.3.2 Confusione e diffusione

Il testo cifrato deve dipendere in modo complesso dalla chiave e dal testo in chiaro, pertanto bisogna creare **confusione** e ciò lo si effettua tramite **sostituzione**.

5.4. CRITTOLOGIA CLASSICA E SICUREZZA AL GIORNO D'OGGI7

Le informazioni contenute nel testo in chiaro devono essere diffuse nel testo cifrato, in modo tale che la modifica di anche un solo simbolo del testo in chiaro si traduca nella modifica di molti, se non tutti i simboli del cifrato (**Diffusione**). La diffusione si crea tramite **trasposizioni**.

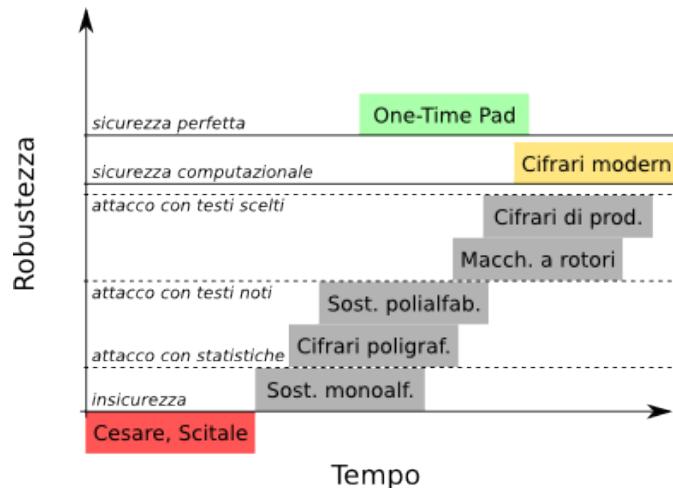
5.3.3 Il cifrario composto

Queste regole sono state introdotte da Shannon che ha indicato anche come conseguirle, cioè ha introdotto il **cifrario composto**. La struttura del cifrario composto alterna stadi di sostituzione (**S-Box**) a stadi di permutazione (**P-Box**).

Shannon ha osservato inoltre che l'esecuzione di più round aumenta la confusione e la diffusione. Oggi tutti i cifrari seguono questo consiglio.

5.4 Crittologia classica e sicurezza al giorno d'oggi

Oggi l'unico cifrario considerato al di sopra della linea di **sicurezza perfetta** è il cifrario **One Time Pad**. Non perfettamente sicuri, ma computazionalmente sicuri sono tutti i cifrari moderni. Completamente insicuri sono i cifrari a sostituzione monoalfabetica e quelli a trasposizione di colonne.



Capitolo 6

Cifrari simmetrici

La crittografia simmetrica studia i **cifrari a chiave segreta**. Tali cifrari vengono usati oggi per proteggere la riservatezza dei documenti, per funzioni di generazione di numeri pseudocasuali, per l'autenticazione e l'identificazione.

I cifrari attualmente in uso hanno quattro qualità:

- **robustezza**: resistono a tutti gli attacchi di crittanalisi finora individuati;
- **velocità**: nelle realizzazioni con hardware special purpose elaborano diversi milioni di bit al secondo;
- **efficacia**: gestiscono stringhe binarie di lunghezza arbitraria;
- **efficienza**: il testo cifrato è praticamente lungo quanto il testo in chiaro.

Vediamo come funziona un meccanismo volto a proteggere la riservatezza di un documento con una chiave simmetrica (figura 6.1):

- il mittente A ed il destinatario B condividono una chiave k_{AB} .
- A deve inviare un messaggio m a B , perciò lo cifra con la chiave k_{AB} ottenendo il testo cifrato c che invierà (l'operazione svolta è $c = E_{k_{AB}}(m)$);
- B riceve c e utilizza la chiave k_{AB} per decifrarlo ottenendo m (l'operazione svolta è $m = D_{k_{AB}}(c)$)

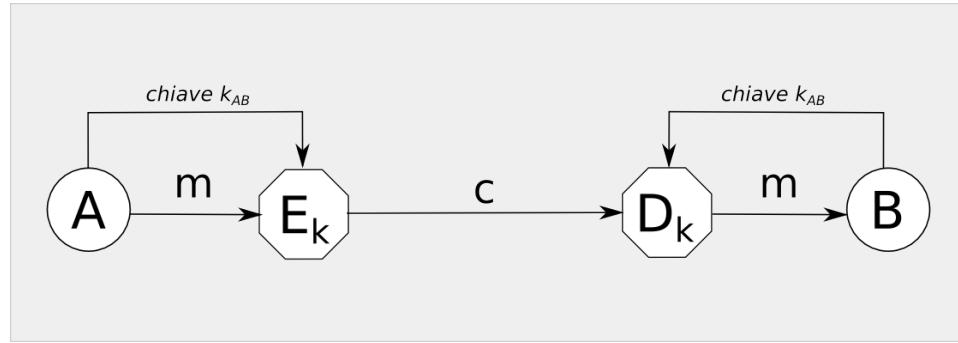


Figure 6.1: Protezione della sicurezza con una chiave simmetrica

In generale m è una parte del testo in chiaro M che A vorrà inviare a B . Proprio sulla natura di m si basa la principale distinzione (non poi nettissima) che viene effettuata sugli algoritmi simmetrici:

- **cifrari a flusso:** sono basati su **one-time pad** e trasformano **uno o pochi bit alla volta**. Sono utilizzati quando occorre proteggere singoli dati generati uno dopo l'altro (ad esempio per una trasmissione seriale di caratteri ASCII, per una chiave WEP...);
- **cifrari a blocchi:** ispirati al **cifrario poligrafico** e al **cifrario composto**, questi cifrano blocchi formati da **molte bit** e risultano molto utili quando occorre proteggere file, trasmissioni di pacchetti ecc.

I cifrari a flusso sono più veloci di quelli a blocchi, tuttavia questi ultimi sono più sicuri.

6.1 Cifrari a flusso

I cifrari a flusso utilizzano una **trasformazione variabile al progredire del testo**. Come abbiamo detto analizzando il **cifrario di Vernam**, questa cifratura può essere realizzata tramite semplici funzioni EX-OR, cioè si può effettuare la somma modulo due fra il bit di partenza ed un bit della chiave. In questo modo si ha dunque che:

- **cifratura:** $c_i = m_i \oplus k_i$
- **decifrazione:** $m_i = c_i \oplus k_i$

La sicurezza di questo metodo sta nel fatto che viene generata casualmente una sequenza di bit aleatori k_i lunga quanto il testo. Non è possibile tuttavia ottenere una sicurezza perfetta dato che i bit di chiave sono pseudo-casuali. Sia in trasmissione che in ricezione si utilizzano macchine a stati finiti dette **generatori di flusso di chiave** per garantire la perfetta sincronizzazione dei due flussi. Con questo strumento si ottengono chiavi periodiche e per garantire sicurezza, il periodo p dev'essere grandissimo. Per rendere imprevedibile un tratto di sequenza utilizzato, gli utenti devono inizializzare i due generatori con un **seme segreto**, di volta in volta diverso.

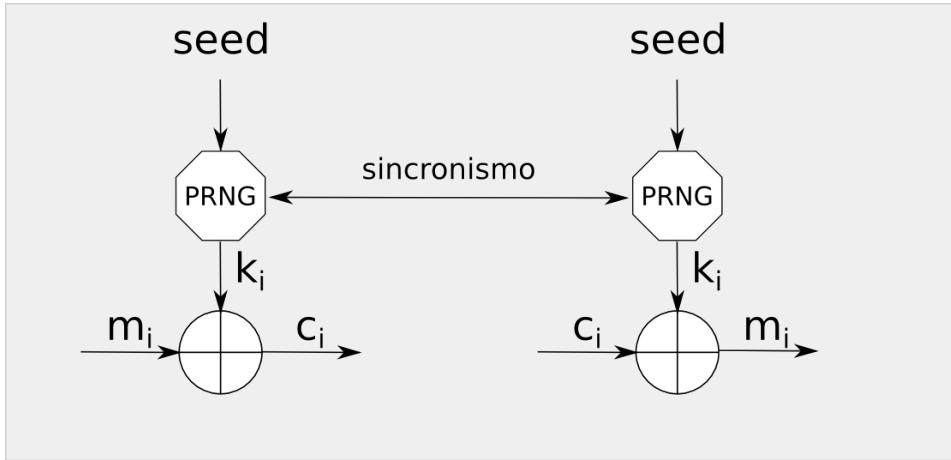


Figure 6.2: Stream Cipher

Dunque ricapitolando, si usa una chiave k_i lunga quanto il messaggio. La sequenza k_i è generata da un generatore di flusso di chiave che ogni volta emette chiavi diverse. Ovviamente le chiavi che si ottengono sono periodiche, quindi è necessario aumentare il più possibile la periodicità e ciò si fa utilizzando un seme segreto, diverso di volta in volta.

Vediamo ora un esempio di utilizzo di cifratura a flusso: il **WEP** *Wired Equivalent Privacy* (figura 6.3). In questo sistema di protezione, si utilizza una chiave k per cifrare i dati. Su una rete wireless possono esserci più utenti connessi, tutti utilizzano la stessa chiave. Ma allora la privacy dove sta? Ogni utente specifica un suo *IV* (vettore di inizializzazione) che viene concatenato alla chiave e usato come *seed* per il PRNG. Il PRNG genera poi la chiave variabile da usare in cifratura. Dato che anche il destinatario deve conoscere il vettore di inizializzazione per poter decifrare il testo, questo viene trasmesso all'inizio, subito prima del testo cifrato. Ad oggi questo protocollo

viene ritenuto insicuro, in quanto la necessità di contenere la dimensione del seed limita la dimensione della chiave.

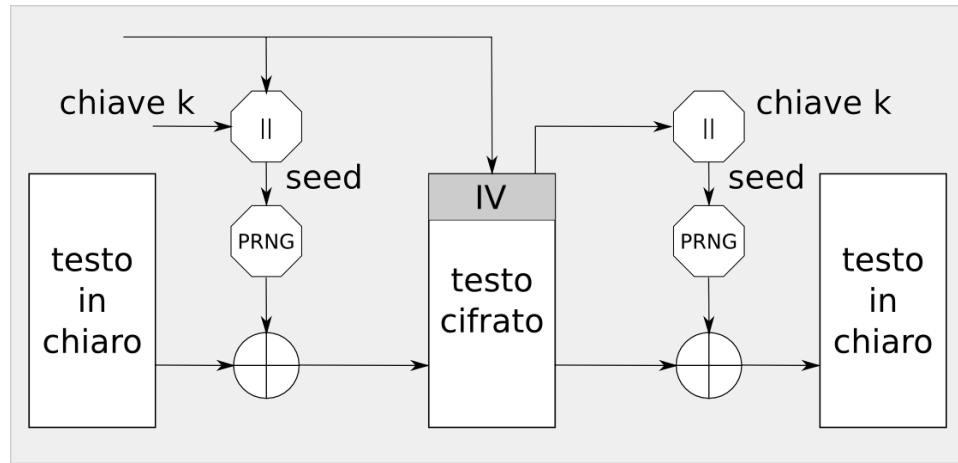


Figure 6.3: WEP

Esistono due differenti tipi di cifrari a flusso:

- **a flusso sincrono**
- **con auto-sincronizzazione**

Nei **cifrari a flusso sincrono**, il flusso dei bit di chiave è generato in modo indipendente dal flusso dei bit di testo; il seed può interessare o la F o la G. Per un corretto funzionamento, i generatori di trasmissione e ricezione devono mantenersi al passo. Nel momento in cui si disallineano, sorgente e destinazione devono far ripartire i loro generatori di chiave e scegliere un diverso punto di inizio della sequenza.

Nei **cifrari a flusso con autosincronizzazione**, il flusso dei bit di chiave dipende dal flusso dei bit di testo cifrato. Una eventuale perdita di sincronismo non richiede intervento da parte degli utenti.

La causa più comune di disallineamento è la **perdita di integrità** del testo cifrato che può essere causata da eventi casuali o attacchi intenzionali.

6.2 Cifrari a blocchi

In questo tipo di cifrari, il testo in chiaro viene suddiviso in blocchi della stessa dimensione aggiungendo alla fine un padding, cioè dei bit di riempimento (se necessari).

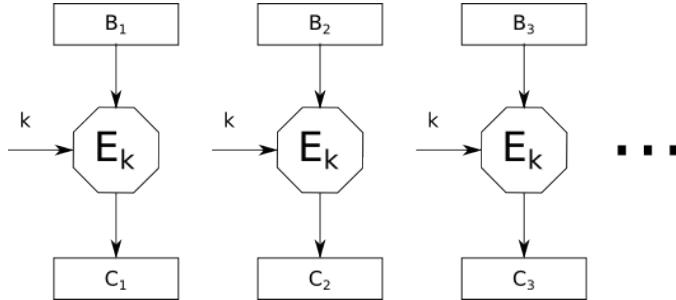


Figure 6.4: schema generale di un cifrario a blocchi

La regola di trasformazione consiste in sostituzioni monoalfabetiche e seguite sulla base della chiave k , o meglio di sottochiavi k_i della chiave k . In figura 6.4 possiamo vedere uno schema generale di funzionamento di un cifrario a blocchi.

Il tempo necessario per rompere il cifrario dipende dalla lunghezza della chiave e dalla potenza della macchina utilizzata.

6.2.1 La rete di Feistel

La maggior parte dei cifrari adottano (con qualche variante) il modello di Feistel che a sua volta discende dal cifrario composto di Shannon.

Il testo in chiaro (lungo $2w$ bit) viene suddiviso in due parti da w bit ciascuna. Queste parti sono dette *vettore di destra* e *vettore di sinistra* e si indicano rispettivamente con R_i e L_i . Ad ogni giro un vettore entra in un blocco F assieme ad una sottochiave k_i (estratta dalla chiave k con un'apposita funzione) ed il risultato viene sottoposto a XOR con l'altro vettore. L'altro vettore è invece libero da elaborazioni. Alla fine di ogni giro i vettori vengono scambiati e riprende l'elaborazione. La funzione F è non lineare.

La rete di Feistel crea **confusione** grazie alle sostituzioni operate da F , la **diffusione** si ottiene dal continuo scambio di R_i e L_i .

Osserviamo che nella rete di Feistel la permutazione è fissa, cioè si ottiene con P-Box priva di chiave.

L'algoritmo di Feistel è uguale sia per cifrare che per decifrare, a patto di invertire le chiavi.

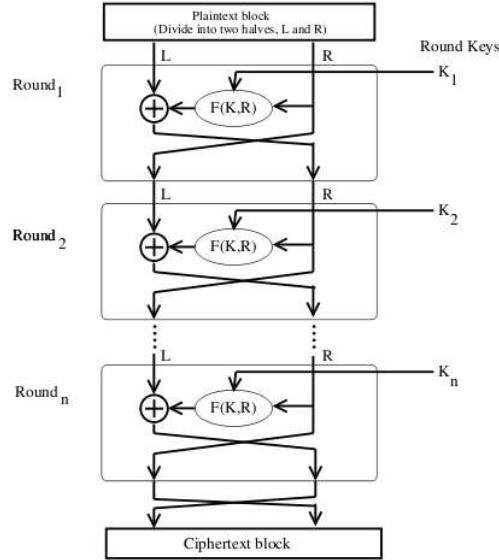


Figure 6.5: La rete di Feistel

6.2.2 DES - Data Encryption Standard

Il cifrario **DES** (*Data Encryption Standard*) fu inventato a metà degli anni '70, commissionato dal NIST e progettato da IBM. Fu progettato per essere usato per circa una decina d'anni, ma fu usato per oltre trenta. Oggi non è più ritenuto molto sicuro per via della lunghezza della chiave.

DES prevede 16 iterazioni, blocchi da 64 bit, ed una chiave di 56 bit (cui ne vengono aggiunti 8 di parità). La funzione F è uno dei punti più delicati della progettazione dei cifrari ed è stata ottenuta ponendo in cascata un'*espansione e permutazione* che porta il vettore da 32 a 48 bit, una *somma modulo 2* del vettore esteso con 48 bit di sottochiavi, una *sostituzione e scelta* che riporta il vettore a 32 bit ed una *permuatazione senza chiave* (P-Box).

Anche la modalità di generazione delle chiavi è una fase delicata.

Per anni si è cercato di rompere il cifrario DES, ma grazie ai vari tentativi si è riusciti a sviluppare due nuove tecniche di crittanalisi:

- **crittanalisi differenziale:** studia come le differenze nei dati forniti in ingresso alla funzione F possono incidere sulle differenze risultanti in uscita.

- **crittanalisi lineare:** è un attacco con testo in chiaro scelto da usare come ingresso di F con cui si cerca di approssimare la trasformazione eseguita dal cifrario con un'equazione lineare da cui dedurre i bit di chiave.

Come abbiamo detto in precedenza, DES oggi non è più ritenuto sicuro. Sono state dunque implementate delle varianti che ora vediamo.

6.2.3 Triple DES

Il **Triple DES** consiste nell'esecuzione di tre volte l'algoritmo DES.

L'implementazione più semplice consiste nell'eseguire tre volte e con tre sottochiavi diverse la cifratura e si indica con *EEE*, mentre un'implementazione che sia maggiormente compatibile con il DES utilizza due cifrature ed una decifratura: *EDE*.

La chiave ha una lunghezza di 168 bit, ma può essere utilizzata una chiave da 112 bit.

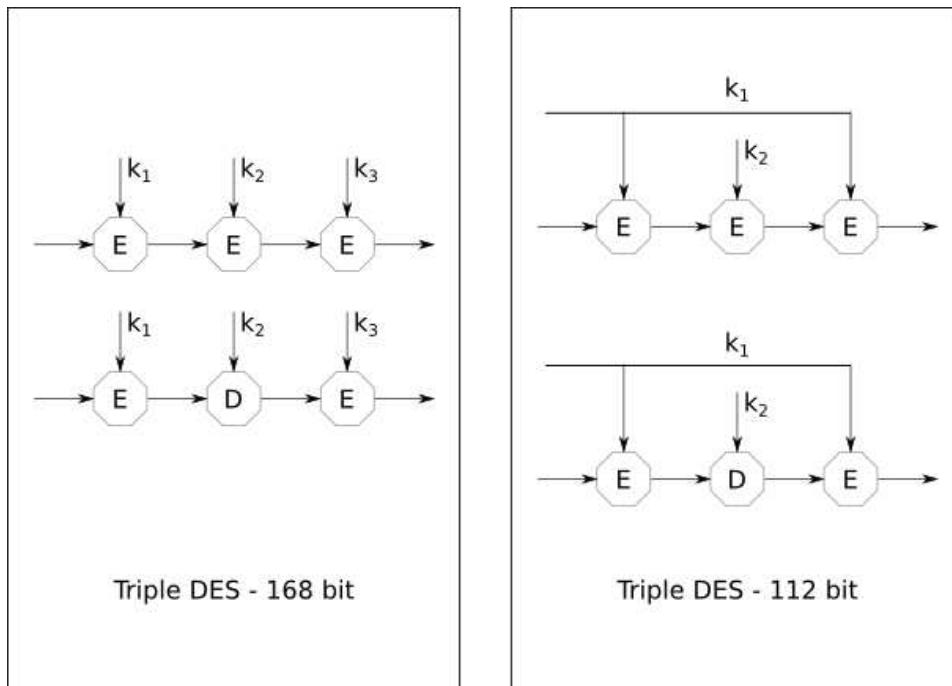


Figure 6.6: Il triplo DES

6.2.4 AES - Advanced Encryption Standard

AES (*Advanced Encryption Standard*), anche detto di Rijndael dal nome del creatore, è stato definito l'algoritmo di cifratura del ventunesimo secolo. È risultato il vincitore di un concorso per la ricerca di un nuovo algoritmo di cifratura indetto dal NIST. AES usa chiavi e blocchi di 128 bit, espandibili per multipli di 32 bit. Tutte le operazioni sono somme a modulo 2 e scorimenti. L'AES non segue lo schema di Feistel, ma uno schema molto più semplice e lineare detto **square**. Ogni round è formato da 4 successive trasformazioni:

- sostituzione di byte;
- permutazione (*shift row*);
- operazioni aritmetiche su $GF(2^8)$ (*mix columns*);
- operazioni di somma modulo due fra i dati in ingresso e la chiave di round (*add round key*).

Attualmente non ha mostrato alcun punto debole ed ha dimostrato di essere facilmente implementabile su tutte le piattaforme garantendo ottime prestazioni e bassa richiesta di memoria (RAM e ROM). AES fornisce un ottimo livello di parallelismo, una buona velocità di key setup ed offre una buona resistenza agli attacchi oggi conosciuti.

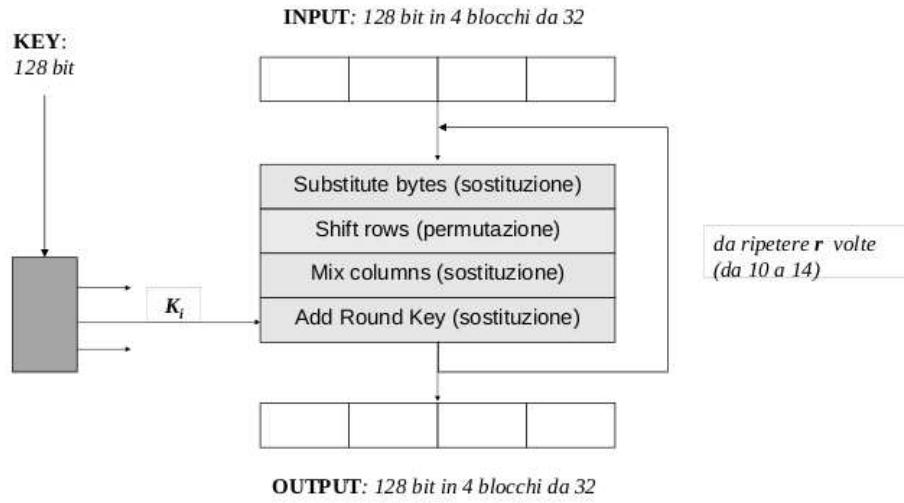


Figure 6.7: L'algoritmo AES

6.3 Modalità di cifratura

La modalità di cifratura di un solo bit alla volta è detta **ECB** (*Electronic Code Book*). Ha il pregio di non propagare gli errori, ma anche una pericolosa vulnerabilità: ogni blocco utilizza la stessa chiave; se ad ogni blocco entra in ingresso uno stesso testo, il risultato sarà identico.

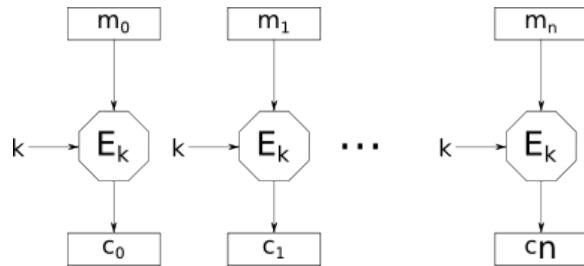


Figure 6.8: ECB

Un esempio di come sfruttare questa vulnerabilità è il seguente: C apre un conto corrente presso le banche A e B . Vuole fare in modo che tutti i trasferimenti da una banca all'altra contengano come destinatario il suo conto corrente sulla banca di destinazione. Come fa? Innanzitutto con un attacco passivo sniffa un trasferimento che lui stesso esegue dalla banca A alla banca B e individua la parte dov'è contenuto il suo conto corrente. In questo modo individua la porzione di messaggio in cui questa informazione è inserita. A questo punto attua un attacco attivo ad ogni successivo trasferimento da un qualsiasi cliente, sostituisce il numero di conto cifrato del destinatario con il suo cifrato e il gioco è fatto.

Per porre rimedio alla vulnerabilità appena presentata, sono state introdotte altre modalità di cifratura che adesso vediamo e che hanno il pregio di far dipendere ogni bit in uscita da ogni bit del testo in chiaro.

Nella modalità **CBC** (*Cipher Block Chaining*), ogni blocco del testo viene dapprima sommato modulo 2 (XOR) all'uscita del precedente blocco di cifratura, poi viene cifrato. In questo modo ogni blocco di testo dipende anche da quelli inviati precedentemente. Per impedire all'intruso di scoprire se vengono inviati due messaggi uguali, si aggiunge un **vettore di inizializzazione** (che viene trasmesso in chiaro alla destinazione).

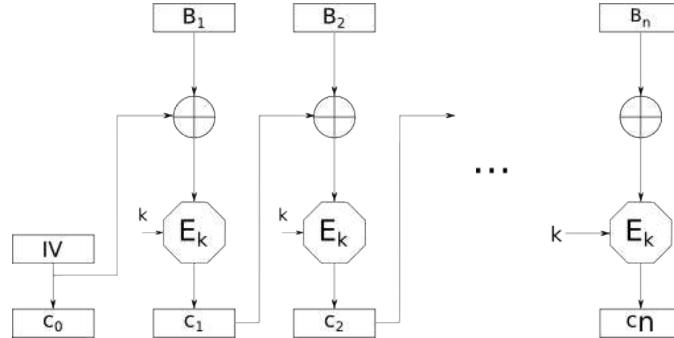


Figure 6.9: CBC

La cifratura **CFB** (*Cipher Feedback*) converte idealmente una cifratura a blocchi in una cifratura a flusso, non effettua padding e opera in tempo reale. L'input della funzione di cifratura è dato da un registro a scorrimento di 64 bit che inizialmente contiene il **vettore di inizializzazione**. I primi n bit più significativi dell'output vengono sommati modulo due (XOR) con i primi n bit del testo da cifrare. Il registro a scorrimento shifta dunque a sinistra di n bit per accogliere negli n bit meno significativi il risultato di questa prima cifratura (che viene mandato anche in output). La cifratura prosegue con altri n bit del registro sommati ad altri n bit del testo da cifrare. La decifratura funziona nello stesso identico modo (anche qui si usa il blocco di cifratura E_k !), ma lo XOR avviene volta per volta con i bit del testo cifrato, anziché con quelli in chiaro. Chiave e registro di scorrimento hanno in pratica il ruolo di generatori di flusso di chiave.

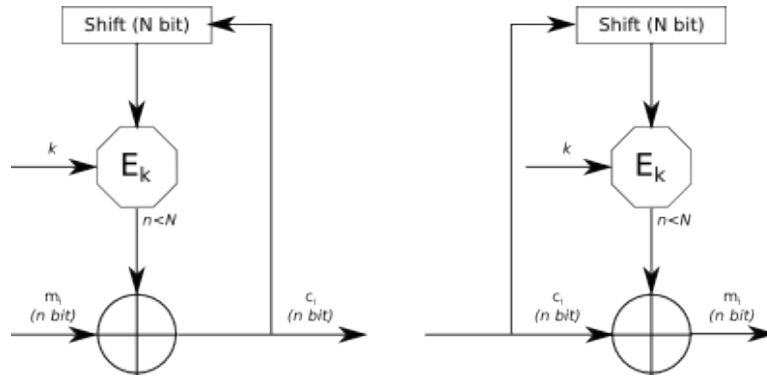


Figure 6.10: cifratura e decifratura nel CFB

La modalità **OFB** (*Output Feedback*) è quasi analoga alla precedente,

con una differenza: ad entrare nello shift register sono gli n bit in uscita dal blocco di cifratura, non quelli in uscita dallo XOR . Anche qui la decifratura avviene allo stesso modo della cifratura e anche qui la funzione di chiave e registro a scorrimento è quella di generare volta per volta la chiave.

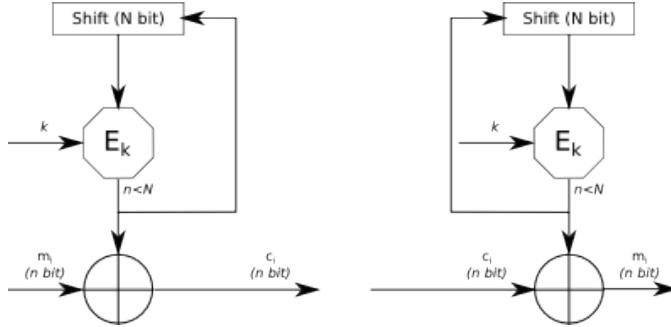


Figure 6.11: cifratura e decifratura nel OFB

Mentre **OFB** implementa un **cifrario a flusso sincrono**, **CFB** realizza un **cifrario a flusso autosincronizzante**. Il vantaggio dell'OFB è di non propagare errori di trasmissione dei bit.

La modalità **CTR** (*Counter*) utilizza appunto un contatore della stessa dimensione del blocco di testo in chiaro su cui operare. Ad ogni blocco deve corrispondere un valore diverso del contatore che viene quindi incrementato ad ogni giro. Il suo valore viene cifrato con la chiave e sommato modulo due al blocco di testo. CTR implementa dunque nuovamente un cifrario a flusso sincrono.

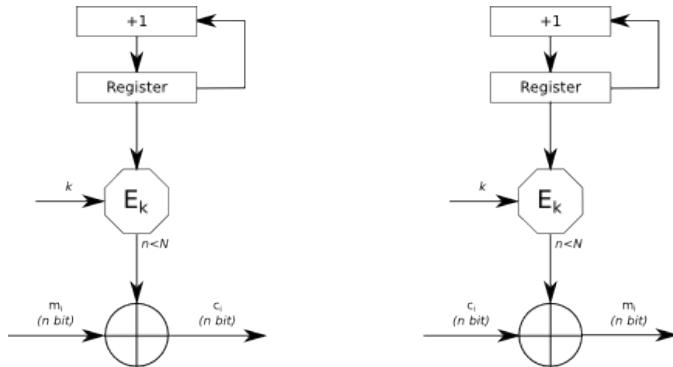


Figure 6.12: cifratura e decifratura nel Counter

6.3.1 Generazione di bit e di numeri pseudocasuali

Le modalità CFB, OFB e Counter possono essere utilizzate per generare sequenze di bit casuali, ovvero come **PRBG** (*Pseudo Random Bit Generator*), mentre come **PRNG** (*Pseudo Random Number Generator*), si utilizzano le modalità ECB e CBC. Esistono vari standard a riguardo, come ad esempio l'**ANSI X9.17** che si occupa della produzione di chiavi e vettori di inizializzazione per il DES.

6.4 Integrità ed origine di un messaggio

Un modo per garantire l'integrità di un messaggio e per avere conferma dell'autore utilizzando la crittografia simmetrica, consiste nel cifrare il messaggio con la chiave condivisa. Il destinatario, se riuscirà a decifrare il messaggio avrà la prova dell'identità. In questo modo si protegge però anche la riservatezza, che in questo caso non ci interessava.

Un metodo alternativo consiste nel fare l'hash con chiave del messaggio, usando ovviamente la chiave condivisa e concatenarlo al messaggio originale. Se il destinatario riesce a ricostruire lo stesso hash con la chiave condivisa, allora integrità ed origine sono provate. L'hash con chiave del messaggio è detto **HMAC** (*Hash Message Authentication Code*). È anche possibile utilizzare una funzione di hash che sfrutta il metodo di cifratura *CBC* ed è la modalità **MAC**, ma le dimensioni del messaggio risulterebbero poi troppo grandi.

Sia nel primo che nel secondo metodo, due problemi rimangono irrisolti: si tratta della possibilità di **ripudio** e di **falsificazione** legato all'uso di una chiave conosciuta da due persone.

6.5 Integrità, origine e non ripudio

Dicevamo nella precedente sezione dei problemi di ripudio e falsificazione. Una **firma digitale** deve possedere 5 requisiti:

- deve consentire a chiunque di poter **identificare univocamente** il firmatario;
- deve essere **non imitabile**;
- deve essere **non trasportabile**;
- deve essere **non ripudiabile**;

- deve rendere **inalterabile** il documento su cui è stata apposta.

Tenendo presente questi requisiti, emerge dunque che una firma digitale garantisce sia l'integrità sia l'origine, sia il non ripudio. L'unico modo per garantire questi tre requisiti facendo ricorso a meccanismi a chiave simmetrica è quello di chiamare in causa una **terza parte fidata**.

Vediamo un possibile schema di firma digitale con terza parte fidata:

1. A invia al **registro atti privati** RAP il messaggio: $A||E_{KA}(A||M)$
2. RAP invia ad A una ricevuta: $E_{KA}(A||T||M||E_R(A||T||M))$. La ricevuta è ottenuta cifrando con un numero generato da un RNG la concatenazione fra l'identificativo di A , un timestamp dell'operazione e il messaggio M . Il RAP avrà nel frattempo salvato nel suo database la voce $A||T||M||R||Firma$
3. A invia a B il documento e la ricevuta: $A||T||M||E_R(A||T||M)$.
4. B interroga RAP per verificare l'autenticità del messaggio ricevuto ed ottiene al passo successivo la risposta: $B||A||T||M||E_R(A||T||M)$
5. RAP comunica a B l'esito della verifica sul messaggio: $E_{KB}(A||T||M)$.

Con questo schema sono stati risolti i problemi di ripudiabilità e di falsificazione, ma ne sono stati introdotti degli altri:

- l'autorità dev'essere sempre online;
- l'autorità non deve costituire un collo di bottiglia;
- l'autorità non deve creare documenti falsi;
- l'autorità deve memorizzare le chiavi in una memoria sicura.

6.6 Gestione delle chiavi

6.6.1 Autorità per la distribuzione delle chiavi

L'obiettivo di un centro di distribuzione delle chiavi è quello di trovare una soluzione più scalabile alla distribuzione delle chiavi perché altrimenti affinchè n utenti possano comunicare fra loro, sarebbe necessari n^2 scambi di chiavi. Il centro di distribuzione delle chiavi è dunque un ente fidato che faccia da intermediario fra coloro che vogliono comunicare.

Prima di vedere come due soggetti A e B possano instaurare una conversazione riservata è necessario assumere che sia avvenuto uno scambio di chiavi fra A e T (l'ente fidato) e fra B e T . Queste sono dette **chiavi master** e vengono utilizzate per comunicare le chiavi di sessione in sicurezza.

Vediamo dunque le fasi necessarie ad instaurare una conversazione riservata (figura 6.13):

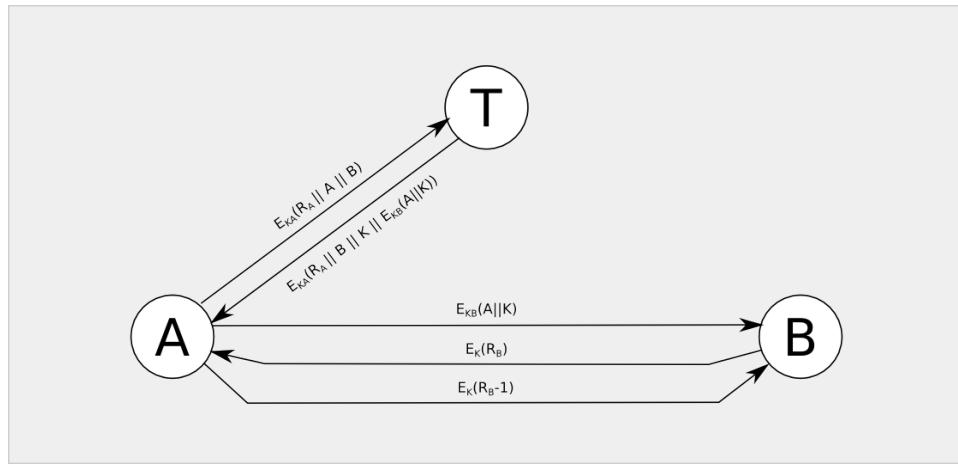


Figure 6.13: Centro di distribuzione delle chiavi

1. A chiede a T una chiave per comunicare con B . Per fare questo invia il messaggio $R_A//A//B$. Il messaggio inviato contiene un numero pseudo-casuale valido una sola volta (un **nonce**) che T dovrà reinviare, l'identificativo del mittente A e l'identificativo del destinatario desiderato (B).
2. T raccoglie la sfida proposta da A , dunque deve reinviare il nonce. Assieme a questo invia anche l'identificativo di B , la chiave generata con un PRNG che A e B dovranno usare per conversare (cioè K), ed un messaggio cifrato con la chiave K_B che A dovrà inoltrare a B per informarlo circa la chiave. Il messaggio di T verso A è dunque $K_A(R_A//B//K//E_{KB}(A//K))$. La cifratura con la chiave condivisa K_A serve a garantire ad A l'identità di T . Inviare il messaggio $E_{KB}(A//K)$ ad A affinché questo lo invii a B è una scelta implementativa che permette di non sovraccaricare il server riducendo le connessioni ad esso.
3. A riceve il messaggio, lo decifra usando la chiave K_A e invia il sotto-messaggio ottenuto da T : $E_{KB}(A//K)$.

4. B riceve il messaggio, lo decifra con la chiave K_B e ottiene la chiave K da usare con A . Potremmo fermarci qui, ma chi assicura a B che A sia chi afferma di essere? Interroghiamolo! Per esser certo dell'identità di A gli invia un nonce: $E_K(R_B)$. Se non facessimo così potremmo subire un attacco, infatti un intrusore potrebbe salvare una copia di un vecchio messaggio $E_K(A//K)$ e riproporlo dopo un po' di tempo dando il via all'algoritmo per come lo vede B . In questo modo invece B vorrà conferma che sta parlando con A prima di inviare direttamente le informazioni.
5. A riceve il messaggio e per provare la sua identità rinvia lo stesso nonce decrementato di uno per evitare attacchi di replica: $E_K(R_B-1)$

Il centro di distribuzione delle chiavi deve comunque attribuire un tempo di vita limitato ad ogni chiave di sessione per garantire una maggiore sicurezza.

6.7 Il protocollo di Diffie-Hellman

Il problema dello schema visto poco fa è quello di essere vincolati ad effettuare lo scambio della chiave fuori banda, o in ogni caso di doversi accordare sulla chiave. Con **Diffie-Hellman** questo problema non c'è.

Lo scambio di chiavi con Diffie-Hellman si basa su uno dei problemi difficili della matematica: **il calcolo dei logaritmi discreti**.

Come funziona il protocollo? Vediamo in quali passi si articola (figura 6.14):

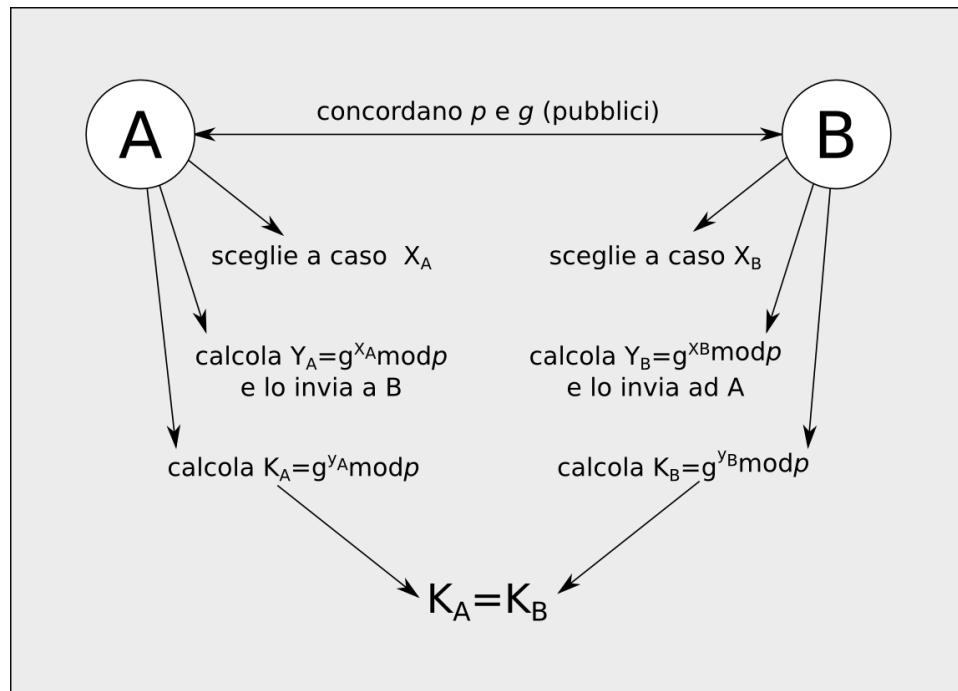


Figure 6.14: Diffie-Hellman

1. A e B devono decidere un **numero primo p** ed un **generatore g** (definiamo generatore di un campo di Galois un numero le cui potenze modulo p sono comprese fra 1 e $p - 1$). p e g sono pubblicamente reperibili, non c'è alcuna segretezza a riguardo.
2. A sceglie a caso un numero X_A , B sceglie a caso un numero X_B . Questi numeri devono restare segreti e devon esser compresi fra 1 e $p - 1$.
3. A a questo punto calcola $Y_A = g^{X_A} \text{ mod } p$, mentre B effettua un calcolo analogo: $Y_B = g^{X_B} \text{ mod } p$
4. A invia a B Y_A , mentre B invia ad A Y_B
5. A questo punto A calcola $K_A = g^{Y_B} \text{ mod } p$ mentre B calcola $K_B = g^{Y_A} \text{ mod } p$. Per le proprietà dell'esponenziazione modulare, $K_A = K_B = K$, dunque K è la chiave.

Un intrusore che conosce p e g (che ricordiamo esser pubblici) può intercettare X_A e Y_A , ma non può scoprire la chiave in quanto ciò sarebbe un problema computazionalmente difficile (complessità esponenziale).

In tutto questo emerge comunque un problema: sebbene il protocollo sia in grado di garantire la sicurezza della chiave, non è garantita ad A l'identità di B e viceversa, perciò bisogna organizzare il sistema in modo tale da risolvere questo problema.

Il protocollo di Diffie-Hellman si rivela migliore della rete mondiale di KDC e del sistema di Incontri&Corrieri in quanto consente di evitare la fase di accordo preliminare, tuttavia non è implementabile in hardware ed è molto molto esoso di risorse.

Parlavamo precedentemente di un problema: rendere sicuro lo scambio di Y_A e Y_B . Una variante è stata pensata e va sotto il nome di **DH/El-Gamal**. In questa variante si fa l'assunzione che se A contatta B , è A ad essere interessato a verificare l'identità di B . Si organizza dunque uno scambio fuori-banda su un canale sicuro di Y_B . Chiaramente così si perde il più grande vantaggio di Diffie-Hellman, ovvero la possibilità di comunicare senza accordi fuori banda... Ma questa è solo una delle varianti possibili.

Le modalità **CFB** (*Cipher Feedback*) e **OFB** (*Output Feedback*) prendono in input blocchi di N bit e anzichè convertirli direttamente, cifrano piccole parti di n bit per volta e li inviano. Dal ricevitore si decifrano anche quattro blocchetti di n bit. La differenza fra i due metodi di cifratura CFB e OFB sta nel fatto che il primo è autosincronizzante, il secondo è sincrono. La cifratura viene realizzata per via dello XOR fra il blocco di n bit e il valore generato dal generatore di flusso di chiave (realizzato con il blocco E ed il registro a scorrimento).

Capitolo 7

Meccanismi Asimmetrici

Nei meccanismi asimmetrici abbiamo una coppia di chiavi per ogni soggetto: una **chiave privata** (usata per firmare o decifrare un documento) ed una **chiave pubblica** (usata per verificare la firma o cifrare un documento). Il problema che studieremo è la progettazione di sistemi per lo scambio di chiavi pubbliche e per il calcolo della chiave privata. La chiave pubblica e quella privata sono legate da una funzione pseudo-unidirezionale. Chiaramente la chiave segreta ha proprietà diverse da quelle di una chiave pubblica, in quanto dovrà essere garantita per lei la massima **segretezza**. La chiave pubblica deve invece poter essere associata univocamente ad un solo soggetto.

Possiamo riassumere le proprietà come segue:

	Chiave pubblica	Chiave privata
disponibilità	•	
autenticità	•	
robustezza	•	•
segretezza		•
integrità	•	•

Chiave privata e chiave pubblica sono legate da una funzione. Tramite una **one-way function** si può ottenere la chiave pubblica dalla chiave privata. L'operazione inversa, trattandosi di una one-way function, non è possibile (computazionalmente). Per la precisione si utilizzano **trapdoor one-way function**, cioè funzioni che appaiono unidirezionali a chi non è in possesso di un segreto. Che si voglia **cifrare** o **firmare** un documento, si userà sempre una **coppia di trasformazioni** in cui un'operazione è semplice e

una è difficile per chi non è in possesso della chiave privata. Nella cifratura è facile cifrare, difficile decifrare senza chiave privata. Nella firma è difficile firmare se non si ha la chiave, è facile verificare la firma.

Gli algoritmi asimmetrici hanno due vantaggi: il primo è che con i cifrari asimmetrici si può ottenere la non **ripudiabilità**, il secondo è che si possono implementare meccanismi di **identificazione attiva**. In realtà il non ripudio si può ottenere anche con la crittografia simmetrica, ma soltanto a patto di coinvolgere una terza entità e con un procedimento più pesante. Lo svantaggio è dato da un'efficienza limitata perché in gran parte si fa uso dell'esponenziazione modulare (operazione molto onerosa).

Lo schema che si utilizza sia per garantire la riservatezza, sia per garantire l'autenticazione, è lo stesso, cambia solo la coppia di chiavi che viene usata:

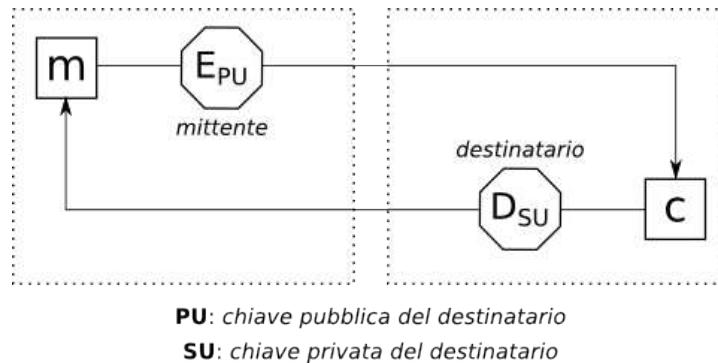


Figure 7.1: Schema di cifratura

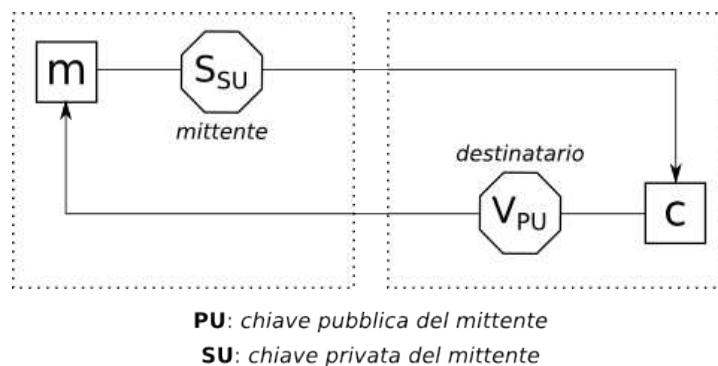
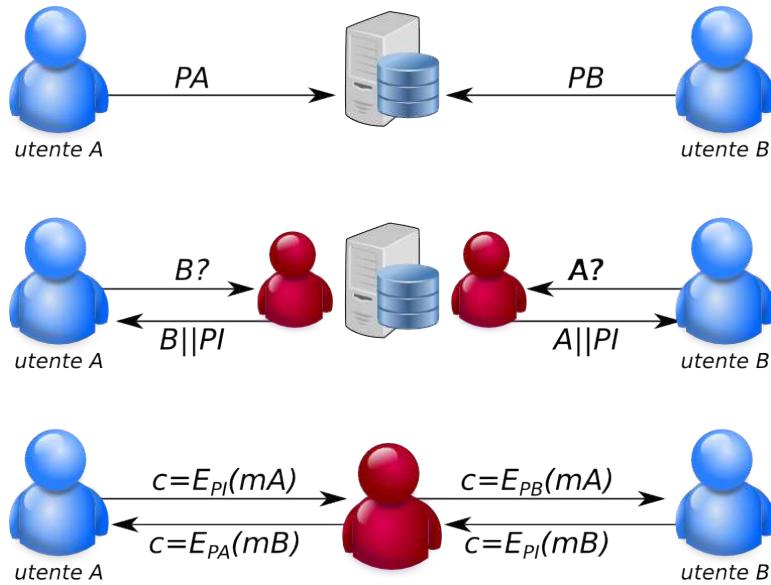


Figure 7.2: Schema di firma

Come faccio ad associare ad una chiave pubblica un proprietario? Se ogni

utente distribuisse autonomamente la propria chiave, dunque in assenza di un ente preposto, si presenterebbe una vulnerabilità ad un attacco di tipo *man in the middle*.



Serve dunque un **ente fidato** per la distribuzione delle chiavi. L'ente provvede a rilasciare per le chiavi dei certificati che garantiscano autenticità ed integrità della chiave. L'utente U che vuole far certificare la propria chiave, dopo aver generato S_u e P_u , contatta l'ente di certificazione e richiede il rilascio del certificato. L'ente, dopo opportune verifiche, rilascia il certificato e lo pubblica.

7.1 Il Certificato

7.1.1 Rilascio del certificato

Vediamo dunque come si genera e usa un certificato:

- l'utente U costruisce ed invia il messaggio

$$mX = X||IX||PX||IPX$$

dove X è l'identificativo univoco dell'utente, IX è un campo di informazioni sull'utente, PX è la chiave pubblica, IPX è un campo di informazioni sulla chiave;

- il sever dice chi è e costruisce una struttura dati m :

$$\begin{aligned} mT &= T \parallel IT \\ m &= mT \parallel mX \end{aligned}$$

- a questo punto l'ente di certificazione deve firmare il messaggio, dunque ne calcola prima l'hash, poi vi appone la firma e lo pubblica:

$$\begin{aligned} H(m) \\ S_ST(H(m)) \\ Cert(PX, T) = m \parallel S_ST(H(m)) \end{aligned}$$

A questo punto il *man in the middle* non è più possibile in quanto l'intruso anche potendo leggere il certificato, non può modificarlo senza che l'alterazione venga rilevata (questo perché non conosce la chiave di firma del server).

Noi depositiamo le nostre chiavi su un server, ma chi ci garantisce che ci posiamo fidare? Lo scopriremo più avanti. Per il momento ci limitiamo a dire che il certificatore può essere o un **ente ufficialmente riconosciuto** o un **qualsiasi utente**. Nel primo caso si segue lo *standard X.509*, il secondo invece è il caso degli utenti di *PGP*.

7.1.2 Formato del certificato

In figura 7.1.2 possiamo osservare il formato del certificato X.509. Il certificato è formato da una parte di dati ed una parte di firma digitale (la parte contrassegnata da *Signature*).

Nella parte destinata ai dati (in cui riconosciamo una sezione relativa al certificatore, un'altra al certificato) abbiamo:

- **Versione:** indica la versione del certificato (quella attualmente utilizzata è la 3, versione che include anche delle estensioni opzionali).
- **Numero seriale del certificato:** identifica univocamente il certificato.

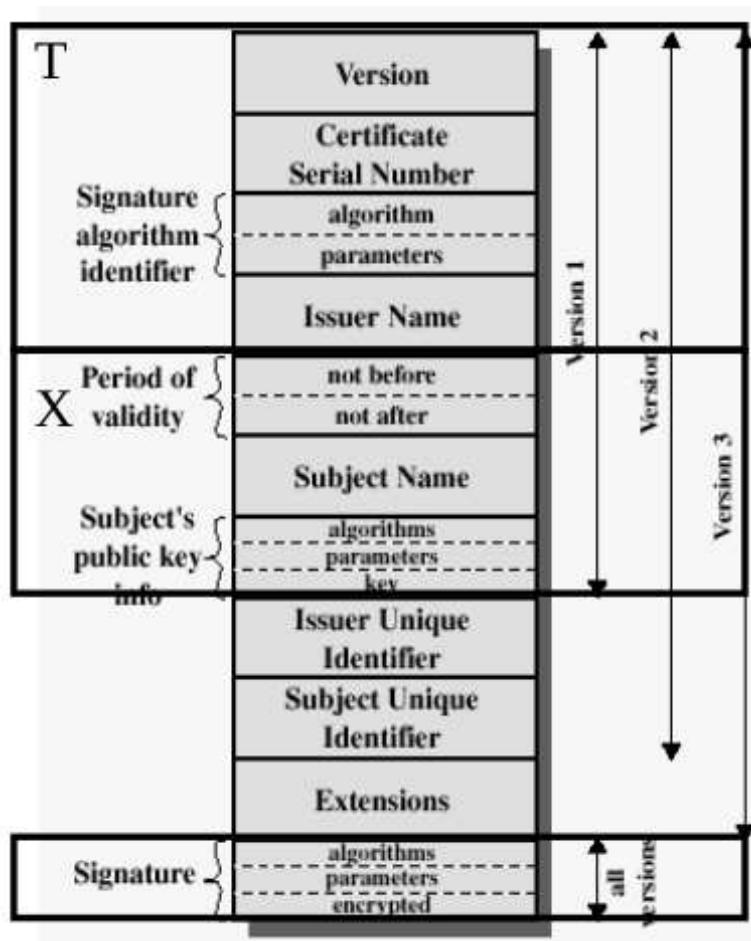


Figure 7.3: Formato del certificato X.509

- **Informazioni sul protocollo di firma:** indica l'algoritmo di firma utilizzato dall'ente di certificazione.
- **Nome dell'ente di certificazione:** identificativo dell'ente di certificazione.
- **Periodo di validità:** è formato dai campi **not before** e **not after** che servono ad indicare la validità del certificato (e dunque il periodo entro il quale l'ente è responsabile).
- **Nome del soggetto:** autoesplicativo...
- **Informazioni sulla chiave pubblica del soggetto:** come e quando è stata generata.

L'ultima parte è la firma del certificato.

In mezzo troviamo le estensioni che sono state aggiunte dalla terza versione del protocollo X.509, ma anche due campi che servono a identificare **univocamente** utente ed ente di certificazione in caso di omonimie.

Tornando alle estensioni, queste possono essere usate per:

- aggiungere informazioni sulla chiave;
- aggiungere informazioni sulle politiche di emissione;
- aggiungere informazioni sull'utente e/o sull'ente di certificazione;
- aggiungere ulteriori informazioni.

Le informazioni aggiuntive per una chiave consentono magari anche di identificare la finalità per cui è stata rilasciata una determinata chiave, infatti si possono rilasciare più chiavi (e di conseguenza più certificati) a seconda dello scenario in cui si intende usare la chiave.

Fra le altre numerose estensioni troviamo il **periodo di utilizzo della chiave privata**. Questo parametro ci consente di preservare le chiavi pubbliche, ma fare in modo che non possano essere emessi nuovi documenti con tale firma. Un'altra estensione che citiamo è quella che consente di specificare un **nome alternativo per il soggetto**, ad esempio un'email, un IP o un sito internet.

7.1.3 La Public Key Infrastructure

Tutto il sistema che consente di ottenere un certificato e pubblicare la chiave va sotto il nome di **Public Key Infrastructure (PKI)** e comprende principalmente tre entità:

- **Registration Authority (RA)**: è l'entità a cui gli utenti si rivolgono per richiedere la certificazione delle chiavi, identificandosi.
- **Certification Authority (CA)**: rilascia il certificato per le chiavi e lo pubblica sul database. Dev'essere il più protetta possibile e dunque non esser contattata dall'esterno e deve poter contattare l'esterno solo in caso di pubblicazione sul DB.
- **Directory (DB)**: ospita i certificati delle chiavi e li mette a disposizione di chi voglia scaricarli.

In fase di registrazione, tutte le entità sono coinvolte, mentre a regime sono coinvolte soltanto l'entità Utente e l'entità Directory.

La Directory

La **Directory** è un database utilizzato per salvare dati di varia entità, ma nel nostro caso faremo riferimento principalmente ai certificati delle chiavi degli utenti. Il principale requisito che la directory deve rispettare è la disponibilità, infatti dev'essere online 24 ore su 24 e sempre accessibile. Per archiviare i dati, la directory segue lo standard **X.500**. Tale standard definisce come salvare e accedere i dati e si basa sul concetto di **entry**. X.500 organizza le entry delle directory in una struttura gerarchica capace di supportare una grande quantità di informazioni. Definisce inoltre potenti metodi di ricerca per rendere più facile il recupero dell'informazione, infatti se un'informazione non è disponibile sul server sul quale fa la richiesta l'utente, la recupera sul server dov'è memorizzata.

Come ritrovo le informazioni all'interno della directory? Le entry all'interno di una directory sono organizzate, come abbiamo detto, gerarchicamente, come in un filesystem. Proprio come se fossero dei file in un filesystem ci si può riferire a loro o facendo uso del **nome assoluto** o del **nome relativo** (a seconda della posizione in cui eseguiamo la ricerca). Il nome assoluto è un **identificativo univoco** dell'utente e va sotto il nome di **distinguish name**. La prima parte del nome definisce la nazione (*C*), poi abbiamo l'organizzazione all'interno di quella nazione che si occupa della certificazione (*O*). Ulteriori livelli nella gerarchia vengono aggiunti con l'indicazione della

località (*L*) e della divisione dell'organizzazione (*OU*). Ad ogni livello possiamo poi avere le foglie, ovvero le entry che indichiamo con *CN*. Segue dunque che il *distinguish name* per una entry sarà di questo tipo: *cn=Bob Aaron, ou=Development, l=Ottawa, o=ABC, c=Canada*.

Ogni entry appartiene ad una **classe** (*object class*). La classe definisce gli attributi collegati ad una data entry (attributi definiti dallo standard).

7.1.4 Richiesta di un certificato

Come fa un utente a richiedere un certificato per una sua chiave? Ogni utente finale sceglie un ente di certificazione che sarà responsabile. Ognuno può avere più chiavi ed enti di certificazione diversi. Il campo della crittografia asimmetrica prevede due diversi meccanismi (come sempre), uno **centralizzato** ed uno **distribuito**, che viene detto **modello a tre parti**.

Concentriamoci al momento sul modello a tre parti. In questo caso abbiamo le tre entità viste precedentemente: utente, registration authority e certification authority (la directory anche è presente ma non la consideriamo come entità).

Nel modello a tre parti, l'utente chiede un certificato alla registration authority specificando alcune informazioni:

- **dati personali**: ogni entità di certificazione sceglie quali dati sono per lei importanti, ma chiaramente avrà bisogno almeno di nome e cognome per identificare un utente;
- **algoritmi usati**: è un'informazione opzionale;
- **chiave pubblica**: chiaramente se l'utente vuole farsi certificare la chiave pubblica... deve inviare la chiave pubblica!
- **firma del messaggio di richiesta**: questa serve per provare di essere in possesso della relativa chiave privata e quindi di non essere un malintenzionato che crea un certificato per una chiave non sua.

7.1.5 Protocolli di gestione

Torniamo ai modelli accennati nella precedente sezione, ovvero allo studio del modello centralizzato e di quello a tre parti.

Nel modello centralizzato abbiamo **due sole entità**: utente e certification authority. La generazione delle chiavi avviene sul server e l'unico messaggio che viene inviato è proprio originato dall'entità di certificazione per comunicare all'utente le chiavi generate per lui. Chiaramente se le chiavi

non vengono generate dall'utente qualcuno oltre all'utente è a conoscenza della chiave privata... Questo è gravissimo e non ci permette dunque di sfruttare il meccanismo di firma digitale e non ripudio. Quand'è allora che si usa questo meccanismo? Lo usano le aziende nel momento in cui vogliono far proteggere ai loro dipendenti i dati su cui lavorano. Questo meccanismo tutela l'azienda, in quanto in caso di licenziamento, se il dipendente impazzisce e cifra tutti i dati per fare un danno all'azienda, quest'ultima può decifrarli conoscendo la chiave...

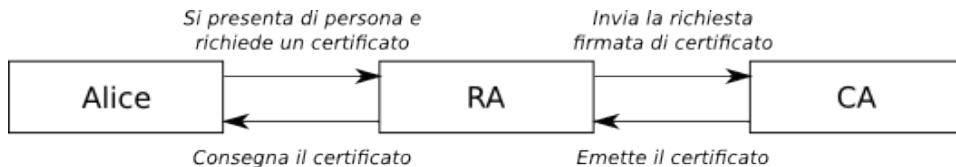
Nel modello a tre parti abbiamo, come dice il nome, **tre entità**: utente, registration authority e certification authority. Qui abbiamo differenti soluzioni che ora vediamo.

- Alice genera le chiavi e si presenta di persona per fornire le sue credenziali fisiche (data di nascita, luogo ecc...) ed elettroniche alla RA. La RA analizza le credenziali e invia le informazioni alla CA che genera il certificato e lo invia alla RA. Quest'ultima lo inoltra all'utente. Come metodo risulta costoso in quanto ogni utente deve possedere una smart card, inoltre ogni utente si deve presentare di persona... Il vantaggio è che la CA è protetta dall'esterno.

In realtà abbiamo due diversi sottoscenari:

- Alice fornisce le sue credenziali e richiede un certificato. La RA le assegna un modulo crittografico che genera le chiavi. Alice genera le chiavi e il modulo crittografico comunica a RA la sua chiave pubblica. RA emette la richiesta per il certificato.
- RA fornisce un modulo crittografico ad Alice e genera le chiavi. RA intanto chiede a CA di rilasciare il certificato e lo consegna all'utente solo quando questo si presenta con le credenziali. Contestualmente Alice riceve la smart card e cambia password in modo tale che nessuno possa accedere alla sua chiave privata.

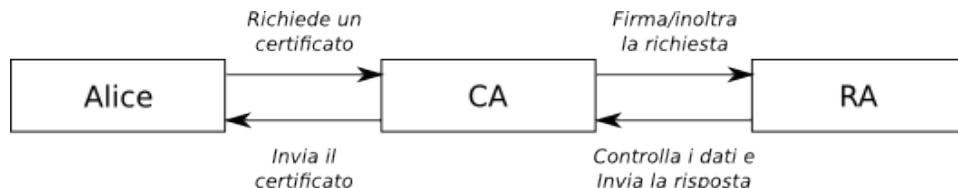
Questo è il metodo utilizzato dalle Pubbliche Amministrazioni in seguito al Decreto Bassanini.



- In questo scenario l'utente presenta le sue credenziali fisiche ed elettroniche all'entità di registrazione ed ottiene un **autenticatore** (ad esempio un PIN) tramite il quale, una volta generate le chiavi (comodamente da casa) potrà inviarle all'entità di certificazione. Abbiamo chiaramente il vantaggio di dover perdere poco tempo dall'entità di registrazione, tuttavia in questo modo l'entità di certificazione è esposta!



- In questo modello l'utente genera le chiavi e richiede un certificato alla CA. La certification authority inoltra la richiesta di validazione alla RA che approva e rimanda indietro la richiesta firmata. La CA a questo punto genera e invia il certificato. Questa è la via meno utilizzata in quanto, come possiamo vedere dalla figura è vero che è richiesto pochissimo carico alla certification authority avendo delegato tutto alla registration authority e che è eliminato il fastidio per l'utente di andare dalla RA, ma è anche vero che la CA si espone terribilmente all'esterno.



7.2 Proof Of Possession

La *Certification Authority* deve avere garanzie circa il possesso della chiave privata da parte del soggetto che richiede il certificato. Rilasciare un certificato senza prova di possesso può permettere vari attacchi. La *POP* è fondamentale ad esempio per garantire il non ripudio.

Possibili scenari in assenza di prova di possesso sono:

- un utente si fa rilasciare un certificato per la sua chiave spacciandosi per un altro utente ed emette documenti sotto falso nome;
- un soggetto può ripudiare la firma.

Il metodo migliore per avere una prova di possesso è il **POP a tempo di firma**: questo metodo consiste nell'inserire nei messaggi un riferimento al certificato e firmare con la propria chiave privata. In questo modo se il destinatario riesce a leggere il certificato ha la prova dell'identità. Questo meccanismo non è attualmente supportato dai protocolli di sicurezza.

La soluzione alternativa consiste nell'emettere il certificato solo se si ha la prova che il richiedente possiede la chiave privata: ad esempio l'utente *A* invia un messaggio di richiesta di un certificato. In esso oltre ad includere la chiave pubblica da certificare, include la versione del messaggio cifrata con la chiave privata. Se la *CA* riesce a decifrare e trova un messaggio identico a quello di richiesta allora può essere certo del possesso. Questo metodo è realizzato tramite il protocollo **PKCS-10**. Altri metodi online consistono ad esempio nell'uso di protocolli challenge-response o nell'invio di un certificato cifrato che in caso di inutilizzo (ad esempio perché chi l'ha richiesto non era davvero in possesso della chiave privata) viene revocato. Questi due metodi valgono però solo per la riservatezza, non per la firma digitale.

Altri metodi sono i metodi **OOB** (*Out Of Band*) in cui la *CA/RA* genera personalmente le chiavi e le rilascia tramite smart card o USB crypto-token. Questi metodi sono però rischiosi in quanto la *CA* mantiene una copia di tutte le chiavi **private**. Come le protegge?

7.3 Revoca dei certificati

Quando si ha il sospetto che la propria chiave privata sia stata compromessa, si deve assolutamente rinunciare ad utilizzarla, ripudiarla e notificarne il ripudio (e chiaramente registrarne una nuova). Il ripudio è l'azione tramite la quale un utente, prima della scadenza, spezza l'associazione fra chiave pubblica e proprietario sancita da un certificato e può avvenire anche per motivi diversi dalla compromissione della chiave privata, ad esempio quando l'utente non ha più il ruolo comunicato nel certificato.

Ok, ma come funziona il ripudio di una chiave? Il ripudio va notificato alla *RA*. La *CA* emette la **revoca del certificato**. Come si fa a notificare gli altri utenti circa la revoca di una chiave? I metodi esistenti vengono classificati generalmente fra **push** e **pull**. Un'altra classificazione è fra **online status checking** e **offline status checking**.

Uno dei metodi più comunemente utilizzati è quello delle **CRL** (*Certificate Revocation List*). Si tratta di un metodo **pull** e che consente la notifica **offline**. Vediamo subito come funziona.

La *Certification Authority* rilascia periodicamente un documento chiamato **CRL** (*Certificate Revocation List*), una lista contenente tutti i certificati revocati. In figura 7.4 possiamo vedere come è composta.

- **Signature Algorithm Identifier:**
 - **algorithm:** algoritmo usato dalla CRL per firmare la lista dei certificati
 - **parameters:** parametri per l'algoritmo
- **Issuer name:** l'identificativo della Certification Authority che ha emesso e firmato la CRL
- **This Update Date:** data di emissione
- **Next Update Date:** data di emissione della prossima CRL
- **Revoked Certificate:** elenco dei certificati revocati. Per ogni certificato abbiamo:
 - **User Certificate Serial #:** numero seriale del certificato revocato;
 - **Revocation Date:** data di revoca;
 - **CRL Extensions:** estensioni...
- **Signature:** comprende i seguenti campi:
 - **algorithms:** algoritmo usato per firmare la lista dei certificati
 - **parameters:** parametri per l'algoritmo
 - **encrypted:**

Volta per volta, la CRL cresce con l'aggiunta dei nuovi certificati revocati. Questo può diventare un problema in quanto le CRL possono diventare molto grandi e dunque onerose da scaricare ed esaminare. Sono dunque state elaborate varie soluzioni:

- eliminare un certificato dalla lista dei certificati revocati dopo la prima apparizione in una CRL;

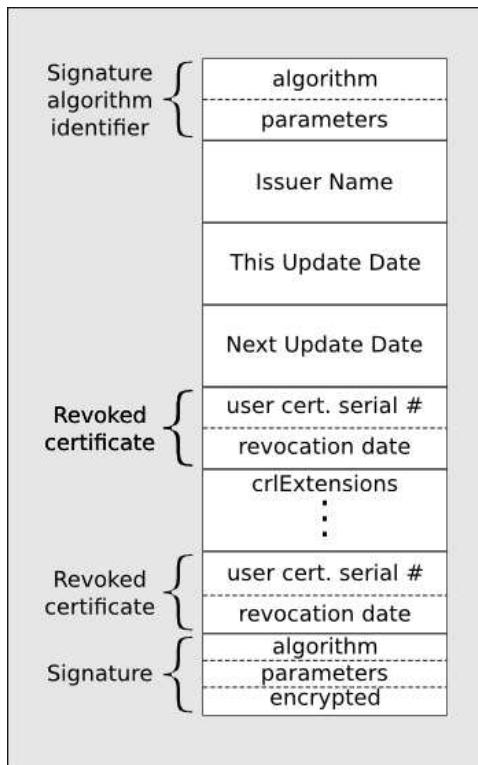


Figure 7.4: una CRL

- pubblicare delle CRL complete e poi soltanto le variazioni (dette *delta CRL*);
- partizionare le CRL in vari gruppi usando dei **CRLDP** (*CRL Distribution Point*).

La prima soluzione non va bene in quanto se un utente vuole sapere se il certificato *X* è stato revocato sarà costretto a scaricare progressivamente all'indietro tutte le CRL finché non trova una corrispondenza o finché non scopre che il certificato non è stato revocato. È dunque molto oneroso per l'utente.

La seconda soluzione non va bene in quanto come la precedente è onerosa ed è necessario per l'utente ricostruire l'intera CRL per avere un quadro completo e vedere se il certificato d'interesse è stato revocato o meno.

La terza soluzione è quella naturalmente usata. Si divide una CRL in più blocchi, partizioni. L'utente che vuole verificare lo stato di un certificato

controlla il campo estensioni dello stesso per verificare il valore del campo **CRLDP**. Questo contiene il nome dell'eventuale lista di revoca all'interno della quale controllare. In questo modo il carico per l'utente è piccolo in quanto deve soltanto scaricare la CRL indicata e verificare al suo interno e si tratterà di una CRL molto piccola in confronto a quella che si avrebbe non partizionando. Un esempio di parametro sul quale basare il partizionamento è quello del *certificate serial number*.

Le CRL vengono emesse con una data cadenza. Fra l'emissione di una CRL e la successiva capita che siano revocati altri certificati, dunque per un periodo è possibile che l'utente non ottenga informazioni aggiornate sulla revoca di un certificato. Questo può essere un problema.

Un rimedio a questo problema è dato dal protocollo **OCSP** (*Online Certificate Status Protocol*), mediante il quale l'utente può avere informazioni in tempo reale circa lo stato di un certificato. È un protocollo pull che funziona online (al contrario delle CRL che una volta scaricate possono essere consultate offline).

I server che offrono il servizio di consultazione in tempo reale dello stato dei certificati possono non essere gli stessi che emettono le CRL, ma anzi essere dei server esterni che fanno uso anche delle CRL per fornire le informazioni agli utenti. I cosiddetti **OCSP responder** seguono principalmente due modelli:

- **Trusted responder:** il server OCSP firma le risposte con una coppia chiave:certificato indipendente da quella della CA per cui sta rispondendo. In genere questo tipo di responder è aziendale o pagato dagli utenti
- **Delegated responder:** il server OCSP firma le risposte con una coppia chiave:certificato diversa in base alla CA per cui sta rispondendo. Questo responder è di solito pagato dalle CA.

Nel valutare i vari schemi che si possono realizzare per informare gli utenti circa la revoca dei certificati, bisogna tener presente alcuni parametri che vediamo racchiusi nella tabella in figura 7.3.

PARAMETRO	PAROLE CHIAVE	
Prestazioni	Lato Amministratore	Picco di Carico e Picco di Richiesta Carico Medio e Richiesta Media Distribuzione del Carico Ritardo Dimensioni
	Lato Utente	Dimensioni Ritardo Massimo Carico Computazionale Banda
Tempestività		Tempo Massimo tra revoca e distribuzione
Scalabilità	Lato Amministratore	Complessità dello schema
Sicurezza		Autenticità Integrità Confidenzialità Non-Ripudio
Standard		Standard Proprietario Teorico Implementato
Espressività		Granularità dell'informazione di revoca
Gestione dello schema	Lato Amministratore	Automatizzato Archiviazione sicura Complessità
On-line vs. Off-line	Lato Amministratore	Frequenza delle connessioni

In quanto a prestazioni, lato utente, ci dobbiamo preoccupare delle **dimensioni** del file da scaricare sia perché ciò richiede una buona connessione e sia perché richiede spazio su un dispositivo di memorizzazione (e considerando che l'utente potrebbe accedere da dispositivi limitati come palmari ciò è un valore importante), del **carico computazionale richiesto** (sempre perché l'utente potrebbe accedere da macchine non troppo potenti), della banda disponibile e quella richiesta. Lato amministratore dobbiamo valutare se siamo in grado di reggere il **picco di carico** e il **picco di richiesta** e valutare a quanto ammontano il **carico medio** e la **richiesta media**. Com'è la **distribuzione del carico**? Il carico è ben distribuito?

Parametro molto importante è poi la **tempestività**, ovvero il tempo massimo fra revoca e distribuzione.

Importante è pure la **scalabilità** dello schema, aspetto che ricade sull'amministratore.

Il livello di **sicurezza** che possiamo garantire qual è? Siamo in grado di garantire **autenticità** delle informazioni passate? E l'**integrità**? Abbiamo bisogno di **riservatezza** nel comunicarle? (È raro ma può accadere che sia necessario...). Siamo in grado di offrire il **non-ripudio** sulle informazioni che comunichiamo?

Che **standard** intediamo utilizzare/progettare? Quali caratteristiche

avrà?

Dobbiamo poi valutare l'**espressività** delle informazioni offerte sulla base della **granularità dell'informazione di revoca**.

Non va sottovalutata la **gestione dello schema** che dev'essere quanto più possibile automatizzato e deve garantire archiviazione sicura dei dati.

Nei vari meccanismi occorre poi valutare la **frequenza delle connessioni**, parametro che può portare a favorire meccanismi offline rispetto a quelli online e viceversa.

7.4 Modelli di fiducia

Non possiamo pensare che tutti gli utenti con cui si vuole comunicare appartengano sempre allo stesso dominio di certificazione: spesso appartengono a domini diversi. Come posso fidarmi di un utente per il quale ad emettere un certificato è stata una CA diversa dalla mia? Bisogna costruire il cosiddetto **cammino di fiducia**.

Il protocollo di ricerca del percorso di fiducia (*certification path discovery*) è articolato come segue:

1. Abbiamo un certificato rilasciato dall'entità CA_X . Troviamo quali entità hanno rilasciato un certificato per la chiave di CA_X .
2. Se fra le entità c'è una *root authority* allora abbiamo trovato il percorso
3. Se non c'è ripetiamo il passo 1 per ognuna delle entità trovate e continuiamo fino ad ottenere il percorso di fiducia.

Facciamo subito un esempio (figura 7.5).

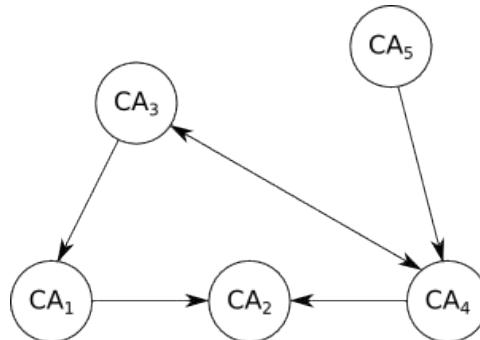


Figure 7.5: Certificate chain

L'utente A invia un messaggio all'utente B . Il certificato di B è stato rilasciato dall'autorità di certificazione CA_5 , quello di A da CA_3 .

L'utente B vuole essere sicuro di potersi fidare del certificato che CA_3 ha emesso per A e cerca dunque quali entità hanno emesso un certificato per la chiave pubblica di CA_3 . Trova CA_4 (con cui fra l'altro c'è fiducia reciproca). Adesso deve trovare quali entità si fidano di CA_4 e trova proprio CA_5 , ovvero la sua CA. Il cammino di fiducia è stato trovato, quindi B si può fidare di A . Il contrario non è possibile in quanto non esiste un percorso di fiducia che permetta di raggiungere CA_5 .

Il fatto che fra CA_3 e CA_4 ci sia una fiducia mutua indica la presenza di **cross-certificates**, ovvero di un certificato emesso da CA_3 per CA_4 e uno emesso da CA_4 per CA_3 .

Trovato un percorso di certificazione è necessario però validarlo. Questo comporta varie azioni come:

- verificare la firma digitale su ogni certificato;
- controllare che il soggetto di ogni certificato sia colui che ha emesso il certificato successivo;
- controllare il periodo di validità dei certificati;
- controllare che i certificati non siano stati revocati;
- verificare le politiche con le quali sono stati rilasciati i vari certificati.

A proposito delle politiche con cui sono stati rilasciati i certificati dobbiamo dire che ogni CA ha le sue. Non esiste uno standard che codifichi univocamente le politiche e di conseguenza queste vengono codificate in maniera diversa da ogni CA. Le politiche vengono inserite nel certificato nel campo estensioni. Solo per i cross-certificate si mettono in piedi dei meccanismi per mappare le politiche di una CA con quelle dell'altra. Sempre nel campo estensioni di un certificato (ma vale solo per le politiche dei cross-certificate), esiste la possibilità di specificare di quali quali politiche dell'autorità che si vuole certificare ci si fida.

In generale il modello di fiducia potrebbe essere **centralizzato** o **distribuito**. Nel primo caso si hanno delle strutture gerarchiche: cioè si ha una **CA radice** che certifica le CA sottostanti.

I nodi foglia sono gli utenti finali. Ad esempio l'utente finale più a sinistra avrà, al termine della richiesta di certificato, il proprio certificato, la chiave pubblica di CA_4 e la chiave pubblica della root CA_1 ! Con un tale modello è facile costruire un cammino di certificazione tra due utenti finali i quali si

rivolgeranno semplicemente alle proprie CA. È una struttura che favorisce appunto il ritrovamento di un cammino di certificazione: basta arrivare fino alla root e poi riscendere fino alla CA di interesse. Garantisce buone caratteristiche di scalabilità e permette di trovare cammini abbastanza corti. L'unico fattore che complica questo modello è la fiducia.

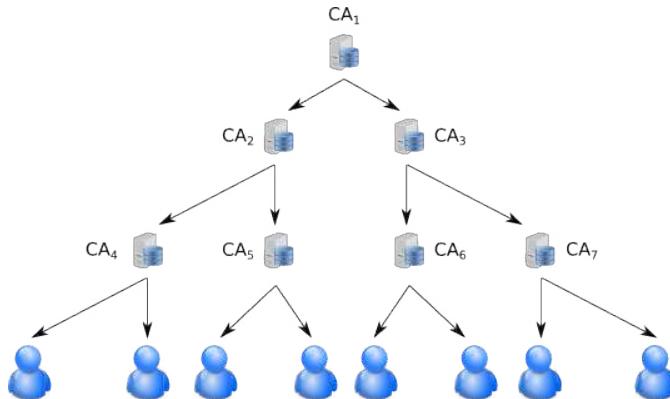


Figure 7.6: General Hierarchical Structure

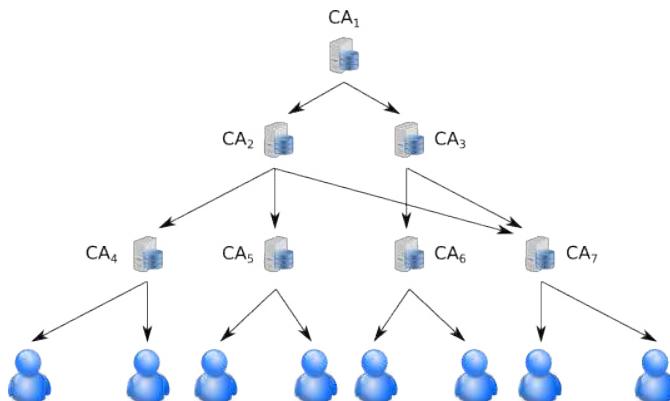


Figure 7.7: General Hierarchical Structure with additional links

In figura 7.7 vediamo che vi è un link aggiuntivo tra CA_2 e CA_7 e uno tra CA_5 e CA_6 . Questi consentono di velocizzare il ritrovamento di un cammino di fiducia senza dover arrivare fino alla root.

Un'altra struttura possibile è la struttura **gerarchica Top-down** in cui i link sono tutti unidirezionali. Ogni fiducia dipende dalla chiave della radice.

Un esempio di modello di fiducia distribuito lo vediamo in figura 7.9

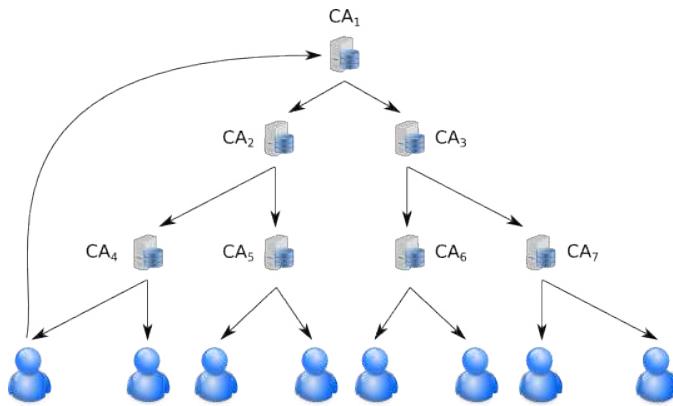


Figure 7.8: Top-down Hierarchical Structure

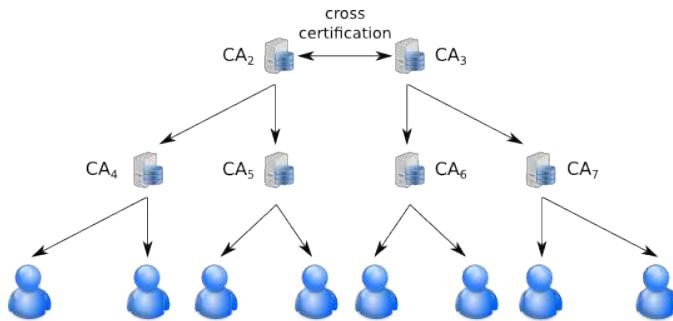


Figure 7.9: Distributed Trust Model

7.4.1 PKI

La **PKI** (*Public Key Infrastructure*) è un'infrastruttura che comprende la **Registration Authority**, la **Certification Authority** ed una **Directory**. Affinché si possa parlare di PKI, devono essere presenti alcune caratteristiche:

- **Key history:** è necessario mantenere una history delle chiavi
- **Cross certification:** deve esistere la possibilità di certificare ed essere certificati da altre CA
- **Key backup & recovery:** è necessario che siano implementate alcune politiche di backup e recovery delle chiavi
- **Aggiornamento automatico della chiave:** sarebbe una caratteristica desiderabile avere l'aggiornamento della chiave automaticamente

alla scadenza

- **Supporto al non ripudio:** è fondamentale garantire agli utenti di poter comunicare certi che sia garantita la non ripudiabilità
- **Timestamping:** ne parleremo in seguito.

La RA dev'essere sempre disponibile, la CA dev'essere rapida anche nella gestione delle CRL. Altri requisiti sono che la CA sia un ente davvero degno di fiducia. Dev'essere poi possibile controllare le CRL per verificare se una chiave è stata revocata o meno.

7.5 Certificati e reti sicure

Richiamiamo per un momento lo schema di Diffie-Hellmann:

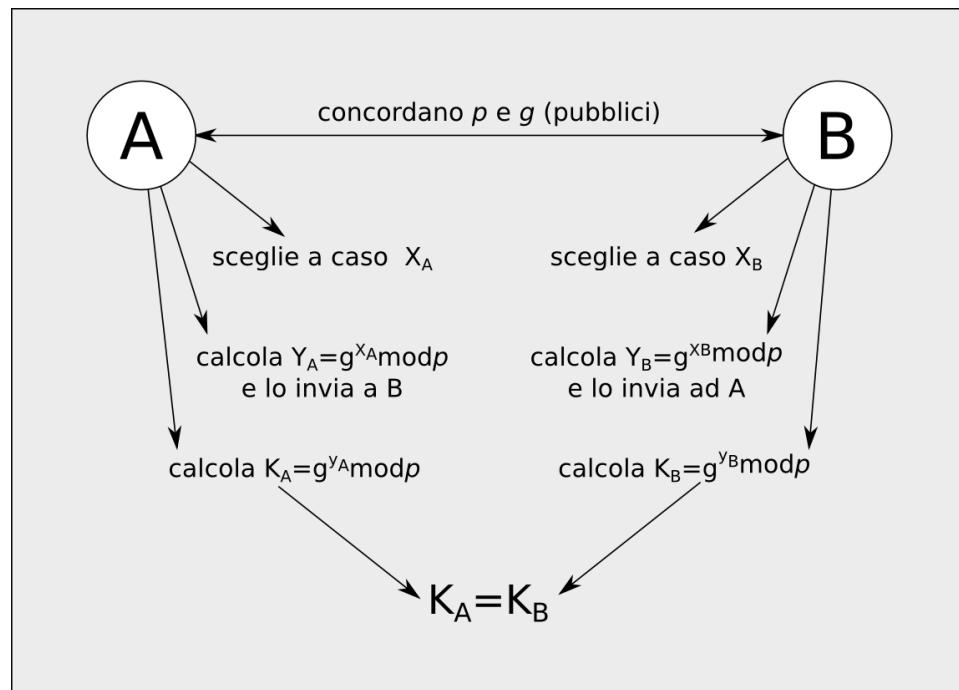


Figure 7.10: Diffie-Hellman

Abbiamo detto che la principale difficoltà è nel garantire che i dati Y_a e Y_b provengano effettivamente da A e B .

Vediamo dunque due accorgimenti che consentono di effettuare uno scambio sicuro di questi due valori.

La prima modalità che vediamo è detta **fixed Diffie-Hellman**. In questa modalità esiste un'entità di certificazione che crea per ogni utente che vuole comunicare tramite Diffie-Hellman un certificato contenente p, g, Y . A questo punto gli utenti si accordano sui parametri da usare per generare la chiave soltanto scambiandosi il certificato. L'algoritmo procede poi allo stesso modo. La chiave che viene generata non è più la chiave definitiva da utilizzare per conversare, ma è una chiave temporanea, detta **pre-master secret**. La chiave definitiva viene generata al passo successivo con una fase di **randomizzazione**, cioè con la concatenazione di due numeri random R_C e R_S concordati precedentemente. Ciò si rende necessario perché ogni utente utilizza sempre lo stesso certificato e quindi questo causa che testi in chiaro uguali provocano pezzi uguali di testo cifrato.

La modalità **ephemeral DH** prevede che all'inizio di ogni sessione di comunicazione, i due partecipanti scelgano a caso il loro dato privato X_i , calcolino il dato pubblico Y_i , e lo autentichino firmando con una chiave privata. Il valore Y_j viene comunicato assieme al certificato della chiave usata per la cifratura. In questo metodo non abbiamo bisogno della randomizzazione, tuttavia questa viene realizzata ugualmente per motivi di uniformità con gli altri standard.

Capitolo 9

Servizi sicuri

I servizi sicuri possono essere sviluppati a livello applicazione, a livello trasporto o a livello rete. Ogni metodo ha i suoi vantaggi ed i suoi svantaggi. A livello applicazione possiamo conseguire qualsiasi requisito (riservatezza, autenticazione e firma digitale) ed i servizi sono di solito flessibili e personalizzabili in quanto sviluppati appositamente per chi li richiede. Esempi di servizi sicuri a livello applicazione sono **Kerberos**, **PGP**, **TSS**. A livello trasporto abbiamo **SSL/TLS**. Implementare servizi sicuri a livelli inferiori a quello applicazione consente di fornire meno visibilità all'utente che risulta dunque meno esposto. A livello rete abbiamo invece **IPsec** che risulta più performante in quanto opera su pacchetti di livello rete. In generale siamo liberi di scegliere se basare il nostro servizio sicuro a livello applicazione, trasporto o rete, ma in un caso la scelta è obbligata: se vogliamo utilizzare la firma digitale dobbiamo ricorrere a servizi a livello applicazione. Analizzeremo dei servizi che integrano uno o più meccanismi di sicurezza ed inizieremo dal timestamping.

9.1 Timestamping

Un documento ha validità legale se è attestata in maniera univoca la data e l'ora in cui è stato firmato. Data e ora sono necessarie per verificare la validità dei certificati impiegati, perciò serve un servizio di timestamping universalmente accettato e non interno al pc di chi firma il documento. Esistono due sistemi di riferimento internazionali: **TAI** (*Tempo Atomico Internazionale*) e **UTC** (*Tempo Universale Coordinato*). In ogni caso utilizzeremo sempre una terza parte che svolge il ruolo di ente fidato.

Una società che offre un servizio di timestamping deve assicurarsi di

rispettare le seguenti proprietà:

1. il tempo della marca non dev'essere falso (cioè l'ente dev'essere un ente fidato);
2. tramite una marcatura dev'essere possibile individuare in modo sicuro un documento, un istante, un autore (dev'essere applicata una firma digitale al documento ed alla marcatura);
3. ogni modifica dev'essere rilevabile anche se effettuata su un solo bit in modo tale da evitare che la marcatura temporale possa essere riutilizzata (ottenibile tramite hash sicuro);
4. dev'essere possibile marcare documenti riservati mantenendone riservato il contenuto (ottenibile tramite hash sicuro);
5. chiunque deve poter far marcare i suoi documenti e deve poter verificare delle marcature. Questo vuol dire che l'ente deve offrire un servizio pubblico e che deve utilizzare una chiave sicura.

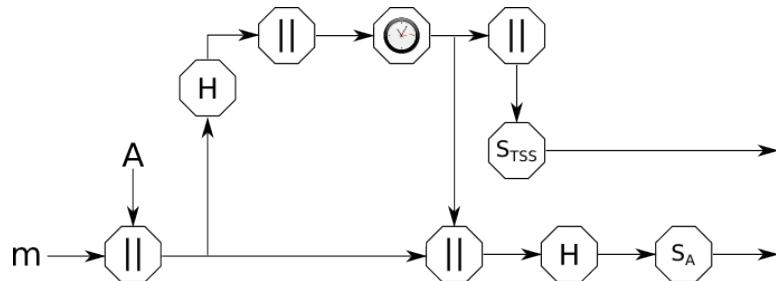


Figure 9.1: Funzionamento di uno schema di timestamping

Descriviamo il funzionamento dello schema. L'entità A produce un messaggio m che vuole firmare digitalmente. Concatena al messaggio il suo identificativo ed utilizza l'hash di $m//A$ per richiedere una marcatura temporale. Utilizza l'hash sia per ragioni di riservatezza che di riduzione del traffico. A $H(m//A)$ viene concatenata la marcatura temporale. Questo messaggio viene firmato dall'entità di certificazione con la sua chiave S_{TSS} in modo che la marcatura non sia ripudiabile. Intanto l'entità A che ha ricevuto la marcatura temporale la concatena ad $m//A$ e calcola l'hash del risultato per poi firmarlo con la sua chiave S_A sottoscrivendo la marcatura temporale ottenuta.

9.2 Kerberos

Kerberos è un protocollo di **autenticazione** che interviene nel momento in cui l’utente richiede un determinato servizio. Kerberos è nato dall’intenzione dei tecnici di offrire servizi di autenticazione, accounting ed auditing, ma ad oggi fornisce solo la prima funzionalità. Kerberos utilizza soltanto funzioni di **crittografia simmetrica** pertanto è più performante, ma adatto soltanto a domini limitati. Kerberos può essere utilizzato per offrire funzionalità di **Single Sign-On**, ovvero per permettere agli utenti un’unica autenticazione per usufruire di più servizi. Kerberos garantisce la mutua autenticazione fra server e client.

Progettando un sistema di autenticazione single sign-on, si possono seguire tre vie:

- mi affido alle applicazioni nelle workstation per autenticare l’utente. I server si fidano delle workstation;
- richiedo che l’utente si autentichi sui server, confidando sul sistema client per la verifica dell’identità degli utenti;
- ogni volta che l’utente accede a un servizio si identifica. Kerberos utilizza questa via. Gli altri metodi richiederebbero costi elevati di gestione e replicazione delle informazioni di autenticazione su ogni workstation.

Quando bisogna progettare un sistema di autenticazione è necessario affrontare tre minacce:

- malintenzionati che cercano di spacciarsi per altri utenti;
- alterazione dell’indirizzo IP;
- intercettazione e riutilizzo dei messaggi.

Vediamo di costruire un sistema sicuro per il single sign-on partendo da un’implementazione semplice ed eliminando pian piano le vulnerabilità.

9.2.1 Semplice dialogo di autenticazione

1. $C \rightarrow AS : ID_C || P_C || ID_V$ - Il client C invia all’ AS (authentication server) il suo identificativo, la sua password e l’identificativo del server V con il quale intende comunicare.

2. $AS \rightarrow C : Ticket$ - L'authentication server invia al client un ticket che nella fase successiva il client spedirà al server cui vuole accedere.
3. $C \rightarrow V : ID_C || Ticket$ - Il client può finalmente comunicare con il server V .

Il Ticket è così composto: $Ticket = E_{kV}[ID_C, AD_C || ID_V]$.

Nel ticket c'è l'indirizzo IP del client (AD_C) che serve a rendere più difficile un attacco di replica. Vediamo però perché questa soluzione **NON** va bene. Innanzitutto salta all'occhio che la password viene inviata in chiaro. Ad ogni connessione ad un server, anche lo stesso server, l'utente deve rifare il protocollo.

9.2.2 Dialogo di autenticazione più sicuro

Bisogna costruire un protocollo che eviti la trasmissione della password in chiaro, e che faccia sì che quando un utente accede ad un servizio non debba rifare sempre tutto il protocollo dall'inizio. Per evitare di dover rifare sempre il protocollo, bisogna introdurre il concetto di **sessione**. C'è un momento in cui l'utente apre una sessione di lavoro, all'interno della quale l'utente può accedere a più tipi di servizi (o anche più volte allo stesso tipo di servizio). Questo vuol dire che bisogna differenziare il protocollo a seconda che si sta aprendo una sessione di lavoro o si è già dentro, a seconda del tipo di servizio che viene richiesto, e se viene richiesto più volte lo stesso tipo di servizio bisogna evitare che l'utente rifaccia tutto quanto. Ha senso autenticare l'utente, identificarlo, al momento dell'apertura della sessione di lavoro, una volta sola.

1. $C \rightarrow AS : ID_C || ID_{tgs}$
2. $AS \rightarrow C : E_{kc}[Ticket_{tgs}]$

$$Ticket_{tgs} = E_{ktgs}[ID_C || AD_C || ID_{tgs} || TS_1 || Lifetime_1]$$

Questo schema di dialogo è più sicuro; quello che avviene è questo: all'apertura di una sessione di lavoro, l'utente contatta l' AS che ha il compito di identificarlo. L'autenticazione non fa uso della password, semplicemente all'utente viene restituito un ticket cifrato con il segreto condiviso fra C e AS in modo tale che soltanto se C è chi dice di essere può utilizzare il ticket. La chiave usata per cifrare non dev'essere in alcun modo salvata nelle workstation. Per far sì che l'utente non debba autenticarsi sempre ogni volta che

accede ad un servizio interno alla sessione di lavoro, introduciamo un'altra entità, il **Ticket-Granting Server** (*TGS*), l'entità che rilascia i biglietti. Quindi l'*AS* identifica l'utente, il *TGS* invece rilascia i biglietti fidandosi dell'identificazione fatta in precedenza dall'*AS*. L'assunzione che facciamo è che ogni utente condivide con l'*AS* un segreto di identificazione, che l'*AS* condivide con il *TGS* un segreto di reciproca identificazione, e chi i server condividano con il *TGS* altri segreti di identificazione.

Il ticket contiene un messaggio cifrato con la chiave che l'*AS* condivide con il *TGS*. Nel ticket c'è l'identificatore dell'utente, l'indirizzo IP dal quale è arrivata la richiesta all'*AS* (e questo è l'indirizzo IP da cui il *TGS* si aspetterà di ricevere la richiesta da quell'utente), l'identificativo del *TGS* con cui l'utente ha chiesto di comunicare, un timestamp (*TS*₁) per indicare l'istante di rilascio del *ticket_{tgs}*, e una validità temporale *lifetime*₁, che indica che l'utente può usare quel biglietto quante volte vuole ma solo finché è valido temporalmente.

Alla fine delle fasi indicate precedentemente **l'utente ha il permesso di parlare con il Ticket Granting Server**. Utilizzerà questo permesso per richiedere l'accesso ad un tipo di servizio di un server "gestito" dal *TGS*. Il permesso è dato da un nuovo ticket, questa volta rilasciato dal *TGS*. Vediamo come il cliente richiede l'accesso a un servizio e come viene fornito il ticket:

1. $C \rightarrow TGS : ID_c || ID_V || Ticket_{tgs}$
2. $TGS \rightarrow C : Ticket_v$

Il ticket del *TGS* è: $Ticket_v = E_{kv}[ID_c || AD_c, || ID_v || TS_2 || Lifetime_2]$

Alla fine di questa fase, l'utente può finalmente accedere ad un tipo di servizio, semplicemente presentando il ticket ottenuto dal *TGS*:

1. $C \rightarrow V : ID_c || Ticket_v$

Per utilizzare lo stesso tipo di servizio, il client non dovrà rifare tutta la procedura di autenticazione. Finché il ticket fornito dal *TGS* non scade utilizzerà quello. Nel momento in cui l'utente vuole accedere ad un altro tipo di servizio gestito dallo stesso *TGS*, semplicemente richiederà un nuovo ticket al *TGS*. Se il ticket fornito dall'*AS* per parlare con il *TGS* è scaduto si dovrà rifare il processo di autenticazione da capo.

Anche questo schema non è esente da problemi:

- se la durata del ticket è troppo breve, l'utente deve re-inserire la password, se è troppo lungo c'è il rischio di intercettazione e riutilizzo.

- se servisse anche l'autenticazione da parte dei server? cioè il client è sicuro di comunicare con il server giusto e quindi di accedere al servizio che ha richiesto? Infine, occorre che il *TGS* dimostri che la persona che utilizza il ticket è proprio quella persona per cui il ticket è stato emesso.
- L'utente che presenta il ticket, è il legittimo possessore? È ancora possibile che un utente malintenzionato aspetti che si disconnetta e riutilizzi il suo pc sperando che il ticket non sia ancora scaduto.

9.2.3 Versione finale

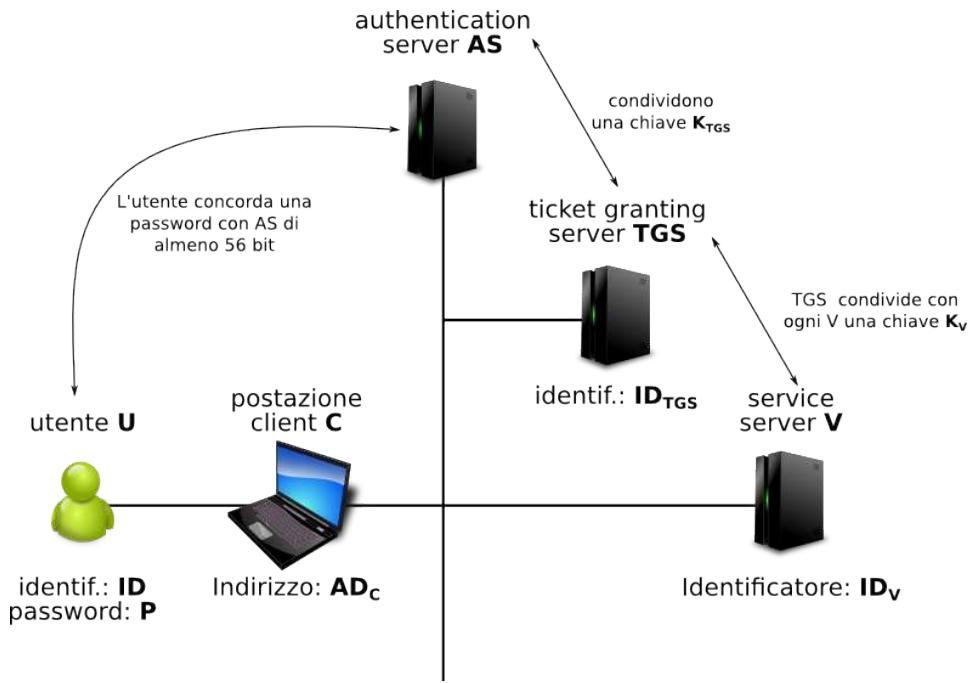


Figure 9.2: Struttura di Kerberos

La versione finale può essere schematizzata così:

1. all'inizio della sessione di lavoro sulla stazione *client C*, l'*utente U* dichiara la sua identità (*ID*) e l'indirizzo del client (AD_C) all'*authentication server AS*:

$$C \rightarrow AS : ID || AD_C || ID_{TGS} || T_1$$

Troviamo inoltre T_1 che è un timestamp e ID_{TGS} che è l'id del TGS con cui vuole comunicare;

2. l'*authentication server* fornisce il permesso d'accesso al *Ticket Granting Server TGS* e lo sfida ad usarlo (cioè glielo manda cifrato con la sua password):

$$AS \rightarrow C : E_P(K_{CT} || ID_{tgs} || T_2 || \Delta T_2 || ticketTGS)$$

Il campo K_{CT} è una chiave che viene restituita all'utente e che quest'ultimo userà per costruire l'autenticatore e dimostrare di essere legittimato alla comunicazione con il *TGS*.

Il ticket *ticketTGS* è costituito in questo modo:

$$E_{KTGS}(K_{CT} || ID || AD_C || ID_{TGS} || T_2 || \Delta T_2)$$

Come vediamo contiene l'ID dell'utente e l'IP della macchina client, nonché l'ID del TGS, un timestamp ed il lifetime (ΔT_2). Il tutto è cifrato con la chiave che l'*AS* condivide con il *TGS*.

3. C risponde alla sfida richiedendo anche l'accesso al *server V*:

$$C \rightarrow TGS : ID_V || ticketTGS || autenticatore_c$$

Il ticket è quello che ha ricevuto dall'*AS*, l'autenticatore (non riutilizzabile) viene costruito dal client con:

$$E_{KCT}(ID || AD_C || T_3)$$

Il *TGS* verifica che i dati contenuti nell'autenticatore siano uguali a quelli contenuti nel ticket (che l'utente non può alterare perché cifrati con la chiave condivisa fra *AS* e *TGS*).

4. *TGS* fornisce a C il permesso d'accesso a *V* e lo sfida ad usarlo:

$$TGS \rightarrow C : E_{KCT}(K_{CV} || ID_V || T_4 || ticketV)$$

Il client riceve una chiave K_{CV} con cui comunicare con il server *V* ed un ticket composto come segue:

$$E_{KV}(K_{CV} || ID || AD_C || ID_V || T_4 || \Delta T_4)$$

Chiaramente questo non potrà essere modificato dal client in quanto è cifrato con la chiave condivisa fra il *TGS* e *V*.

5. C si qualifica a *V*:

$$C \rightarrow V : ticketV || autenticatore_C$$

L'autenticatore viene costruito in questo modo:

$$E_{KCV}(ID_C || AD_C || T_5)$$

6. V si fa identificare da C :

$$V \rightarrow C : E_{KCV}(T_5 + 1)$$

7. C usa il servizio. Se ha bisogno nuovamente da un servizio da V torna a 5, altrimenti se deve effettuare l'accesso a un nuovo server va al passo 3. Dopo N secondi scade il tempo.

9.2.4 Accesso a servizi di un altro dominio

Come si fa ad accedere a servizi che si trovano in un altro *realm* (dominio)? L'accesso a tali servizi può avvenire solo nel momento in cui esiste un **rapporto di fiducia** fra il dominio di appartenenza del client e quello del servizio destinazione.

I *TGS* dei vari domini devono condividere delle chiavi simmetriche, mentre la password dell'utente deve restare nel dominio dell'utente, nota solo al suo *AS*.

Dunque il client richiede al proprio *TGS* un ticket per un servizio. Se tale servizio è locale viene fornito direttamente, altrimenti se fa parte di un altro dominio costruisce un ticket tramite il quale C possa presentarsi al *TGS* dell'altro dominio. A questo punto è il *TGS* del dominio di destinazione a costruire un ticket tramite il quale C possa accedere a V .

Se esistono n ticket granting server, esisteranno n^2 chiavi dal momento che non vi è un centro di distribuzione delle chiavi.

9.2.5 Kerberos V4

Di Kerberos esistono attualmente due versioni (4 e 5). La versione 4 ha delle limitazioni:

- forte dipendenza dal DES che prevede 56 bit di chiave, mentre ora ne servono almeno 128;
- dipende dallo schema di indirizzamento IP;
- il dimensionamento della durata del ticket porta a compromessi tra robustezza ed efficienza;
- problema della scalabilità: è adatto a domini limitati.

9.3 PGP

PGP (*Pretty Good Privacy*) è un protocollo ideato da Philip Zimmerman nel 1993 per fornire autenticazione e riservatezza in ambito mail e file storage.

PGP fornisce in realtà 4 servizi avvalendosi di 14 meccanismi.

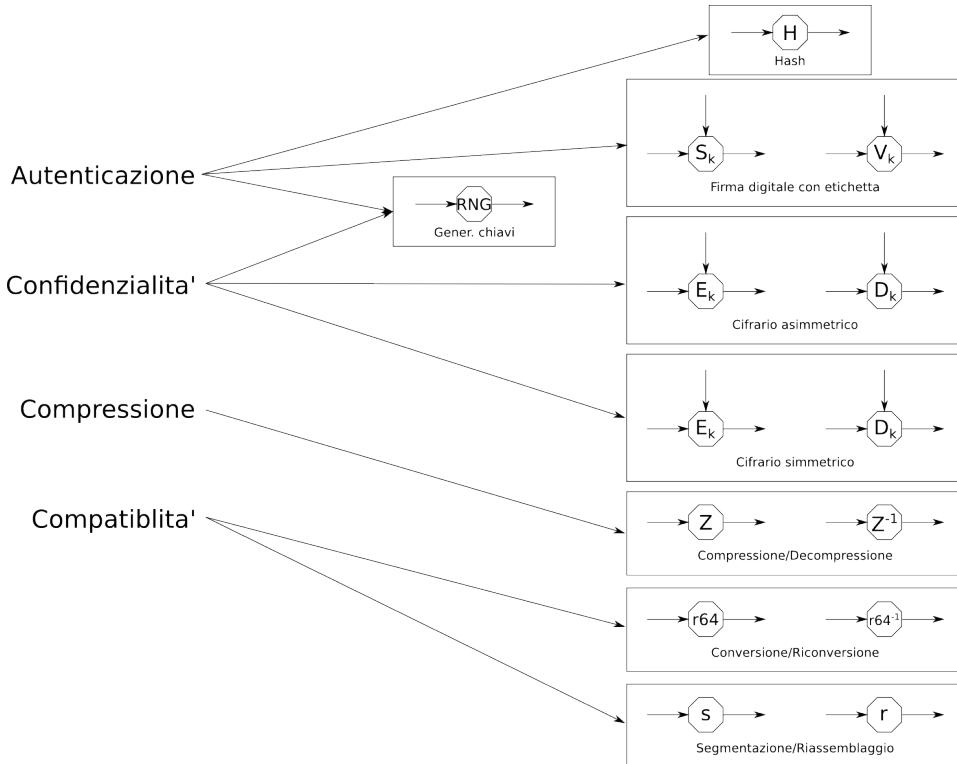


Figure 9.3: Servizi e meccanismi di PGP

Vediamo quali sono e come funzionano i servizi offerti:

- **Autenticazione:** verifica dell’identità di un’entità. Questo servizio viene offerto facendo ricorso a 4 meccanismi:
 - **RNG:** viene utilizzato uno pseudo random number generator per generare le chiavi da utilizzare;
 - **Hash:** si utilizza l’hash per calcolare il digest del messaggio in quanto è su esso che viene apposta la firma;
 - **Firma e Verifica:** i blocchi S_k e V_k servono a firmare e decifrare i dati.

- **Confidenzialità:** mantenere riservato il contenuto di un messaggio a tutti tranne a chi è autorizzato a leggerlo; questo requisito è ottenuto tramite:
 - **RNG:** per generare le chiavi;
 - **E_k:** per cifrare;
 - **D_k:** per decifrare.

E_k e D_k possono essere metodi di (de)cifratura simmetrica o asimmetrica. In particolare, di solito si usa la cifratura simmetrica per cifrare lunghi messaggi. In questo caso la chiave viene generata da un RNG e scambiata tramite crittografia asimmetrica.

- **Compressione:** riduzione della dimensione dei dati; la compressione viene di solito utilizzata in combinazione con le funzioni per ottenere riservatezza ed autenticazione. In particolare si utilizza l'algoritmo **ZIP** per comprimere/decomprimere i dati dopo la firma e prima della verifica nel caso dell'autenticazione o prima di cifrare e dopo aver decifrato nel caso della riservatezza. Questa differenza riguardo al momento in cui viene applicata la compressione è relativa al fatto che è preferibile avere il messaggio subito disponibile. Comprimere prima di applicare la firma, vorrebbe dire che poi in fase di verifica della firma (che può avvenire anche in tempi molto lontani) bisogna ri-comprimere il messaggio per calcolarne il digest e firmarlo, oppure bisogna mantenere sia la versione firmata, sia il testo in chiaro (e questo nel caso soprattutto poi di firme multiple diventerebbe scomodo). Nella riservatezza si preferisce cifrare prima perché aumenta le performance e riduce le ridondanze.
- **Compatibilità con email:** PGP offre trasparenza alle applicazioni email. I testi cifrati vengono convertiti in ASCII usando la conversione **Radix-64**. Radix-64 si occupa anche di gestire la **segmentazione** dei messaggi (in quanto questi non possono essere più lunghi di 65.000 byte). Il protocollo Radix-64 funziona applicando un'estensione di ogni gruppo di 3 bit in 4 bit causando perciò un'espansione del 33% a cui si rimedia tramite compressione.

Tramite PGP è possibile anche ottenere autenticazione & riservatezza. Come si può fare? Si genera una chiave di sessione k , monouso. Si firma il messaggio m con la propria chiave privata S_A , si cifra il messaggio e la sua firma con la chiave k , si cifra la chiave k con la chiave pubblica del destinatario

P_B e si invia il tutto. Il destinatario decifrerà la chiave con la sua chiave privata S_B ed utilizzerà la chiave k ottenuta per decifrare il messaggio con la sua firma che verificherà usando la chiave pubblica del mittente P_A . In figura 9.4 è possibile vedere uno schema riepilogativo

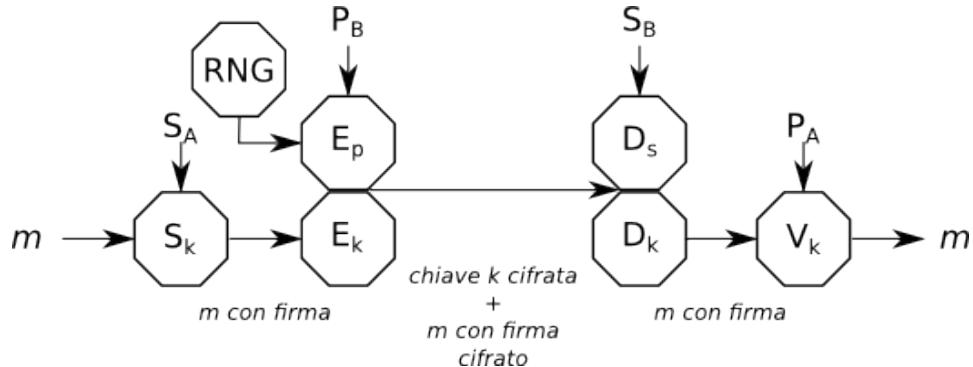


Figure 9.4: Autenticazione e riservatezza con PGP

Cosa succede nel caso si voglia inviare un messaggio riservato a più destinatari? Si creano n versioni cifrate della chiave k con le chiavi pubbliche dei rispettivi destinatari e si concatenano tutti i pezzi così ottenuti per formare il messaggio da inviare.

La modalità di cifratura usata da PGP è la **CFB** (*Cipher Feedback*).

In **PGP** ogni utente ha due portachiavi, uno **pubblico** ed uno **privato**. Nel portachiavi privato, l'utente ha le sue chiavi private (possono essere più di una), nel portachiavi pubblico vi sono invece le proprie chiavi pubbliche e le chiavi pubbliche delle entità con cui comunica. Vediamo la struttura dei due portachiavi:

Portachiavi privato:

Possiamo vedere il portachiavi privato come una tabella costituita dalle informazioni sulle chiavi private, informazioni che vediamo racchiuse nella tabella seguente:

Timestamp	Key ID	Public Key	Encrypted Private Key	User ID
-----------	--------	------------	-----------------------	---------

- **Timestamp:** è l'indicazione del momento in cui la chiave è stata generata. Non è un timestamp universale.
- **Key ID:** ogni utente può avere più chiavi. Nel momento in cui un

utente *A* invia un messaggio firmato ad un utente *B*, come fa a dirgli quale chiave utilizzare per verificare l'autenticità del messaggio? Non gli comunica l'intera chiave perché sarebbe dispendioso, ma un ID. L'ID di una chiave è costituito dai 64 bit meno significativi. La probabilità di errore (cioè di identificare più di una chiave fra quelle di un utente con lo stesso ID è bassissima).

- **Public Key:** la chiave pubblica associata alla chiave privata.
- **Encrypted Private Key:** la chiave privata non viene memorizzata in chiaro, ma cifrata utilizzando una password che viene chiesta all'utente.
- **User ID:** identificativo dell'utente.

Portachiavi pubblico:

Un portachiavi pubblico è invece così composto:

Time-stamp	Key ID	Public Key	Owner Trust	User ID	Key legitimacy	Signatures	Signature Trusts
------------	--------	------------	-------------	---------	----------------	------------	------------------

Analizziamo i campi (nuovi) nel dettaglio:

- **Owner Trust:** fiducia iniziale che io ripongo in una data chiave pubblica. Esistono diversi livelli di fiducia: contenuto fidato, non fidato, parzialmente fidato... A seconda di come ho ottenuto la chiave e di quanto conosco il proprietario posso impostare il livello di fiducia. Esempio: Bob mi ha passato la sua chiave dal vivo, io lo conosco, mi fido di lui, dunque appongo "fidato" sulla sua chiave.
- **Key Legitimacy:** PGP calcola automaticamente sulla base di *Owner Trust* e *Signature Trust* il livello di fiducia.
- **Signature(s):** la chiave pubblica può ad esempio essermi stata fornita tramite un certificato che è una struttura dati firmata. Se il certificato è X509 è emesso da un'entità di certificazione. Nel caso dell'esempio precedente in cui Bob mi ha passato a mano la sua chiave, il campo Signature (e con esso il campo collegato *Signature Trust*) rimane vuoto.
- **Signature Trust(s):** fiducia nel firmatario del certificato (vedi campo precedente).

Bisogna stabilire un livello di fiducia. questo non è mai un livello certo perché non c'è un'entità centrale che si occupa di certificare un utente, ma la fiducia è sempre data da una componente soggettiva (la fiducia che io pongo in una chiave) ed una componente oggettiva (la fiducia che mi deriva dagli altri utenti che certificano una chiave). Nel tempo il livello di fiducia della chiave di un utente si definisce in maniera più precisa acquisendo informazioni da altri utenti.

Anche nel caso di revoca, se io revoco una mia chiave, è necessario propagare agli utenti tale informazione. Il processo potrebbe essere lento.

Capitolo 10

Servizi sicuri per la comunicazione in rete

I servizi sicuri possono essere implementati sia a livello applicazione, sia a livello trasporto, ed anche a livello rete. Abbiamo già visto alcuni protocolli di livello applicativo (*timestamping, Kerberos, PGP*), in questo capitolo ne studiamo invece altri ad altri livelli.

A livello applicativo non si ha trasparenza verso l'utente, in quanto è necessario installare nuove applicazioni (a cui gli utenti dovranno essere educati), ma proprio questo punto fa sì che i servizi siano altamente personalizzabili.

10.1 SSL/TLS

SSL/TLS ha come obiettivo quello di fornire autenticazione d'origine e riservatezza dei dati inviati all'interno di un'applicazione. In realtà parliamo di una suite di protocolli, alcuni collocati a livello applicativo, altri a livello di trasporto. Questo è il motivo per cui SSL viene di solito etichettato come un protocollo di livello trasporto.

Importanti sono i concetti di **sessione** e **connessione**. Con sessione si intende un'associazione logica fra client e server. La sessione viene creata dal protocollo di handshake e definisce un insieme di parametri crittografici che possono essere condivisi fra molteplici connessioni. All'interno di una sessione possiamo infatti avere molteplici connessioni fra client e server per molteplici scambi di informazioni. Avviare più connessioni all'interno di una sessione evita la costosa rinegoziazione dei parametri di sicurezza.

2CAPITOLO 10. SERVIZI SICURI PER LA COMUNICAZIONE IN RETE



Lo stato di una sessione (che corrisponde allo stato di ciascun endpoint della sessione) è rappresentato dalle seguenti informazioni:

SSL/TLS fanno uso di due strutture dati per tenere traccia dello stato di sessioni e connessioni.

Stato di una sessione:

- **identificatore di sessione:**
- **certificato X.509 del peer:** se il client ha negoziato una sessione di lavoro con un server, il client ha identificato il server tramite certificato elettronico.
- **metodo di compressione:** il metodo con cui il server comprime i dati;
- **cifrario ed algoritmo hash:** vengono memorizzate le informazioni sul cifrario e l'algoritmo di hash utilizzati;
- **segreto principale (master secret):** ogni client e ogni server hanno stabilito a priori un segreto condiviso che verrà usato all'interno delle singole connessioni per produrre le chiavi di sessione.

Stato di una connessione:

- **Random number:** per evitare attacchi di replay. vengono usati nel d-h per randomizzare il numero per il calcolo del segreto condiviso. se c e s decidono di usare dh devono poter scambiare in numeri random che permettono di randomizzare ogni volta il segreto in maniera corretta.
- **Server write MAC secret:** segreto usato per il MAC dei dati del server.
- **Client write MAC secret:** segreto usato per il MAC dei dati del client.

- **Server write key:** chiave condivisa per la cifratura lato server e de-cifratura lato client.
- **Client write key:** chiave condivisa per la cifratura lato client e de-cifratura lato server.
- **Vettore di inizializzazione:** può servire per la cifratura se si usa ad esempio la modalità di cifratura CBC.
- **Sequence number:** per ogni messaggio trasmesso e ricevuto viene memorizzato da entrambi il numero di sequenza.

Come si instaura una sessione? Seguendo alcuni passi:

1. **Fase 1:** la comunicazione inizia con un messaggio di *hello* da parte del client, cui risponde il server con un analogo messaggio. In questi due messaggi client e server comunicano ciò che sanno fare. Il contenuto dei messaggi *hello* include la versione di SSL supportata dalle parti, un random number (usato per lo scambio delle chiavi ad esempio con Diffie-Hellman), ID di sessione, suite di cifratura supportata, metodi di compressione supportati. Questa fase consente dunque di negoziare la suite di cifratura e di compressione da utilizzare.
2. **Fase 2:** il server si autentica presso il client inviando il proprio certificato. Il certificato viene controllato dall'applicazione del client, ma è **importantissimo** sottolineare che le applicazioni oggi esistenti verificano solo la validità temporale del certificato e l'integrità dello stesso, ma **non** se è stato revocato, dunque SSL va bene se il requisito del non-ripudio non è essenziale;
Qui avviene lo scambio delle chiavi che permette a regime di cifrare e avere autenticazione;
3. **Fase 3:** in questa fase avviene l'accordo sul segreto ed eventualmente l'autenticazione del client presso il server (non obbligatoria). Se il server ha bisogno che il client si autentichi invia una *certificate request*. Al termine di questa fase il segreto è stato condiviso;
4. **Fase 4:** chiude il protocollo e controlla che tutti i dati stati inviati correttamente e integralmente (solitamente con funzioni hash crittografiche). A questo punto tocca al protocollo **record** proteggere i dati inviati.

4 CAPITOLO 10. SERVIZI SICURI PER LA COMUNICAZIONE IN RETE

Stabilita una socket fra client e server (che segue ad una fase di mutua identificazione) viene garantita l'autenticazione dell'origine e riservatezza. Terminato questo protocollo è possibile instaurare le connessioni per inviare dati.

Vediamo quali sono i protocolli di cui fa uso *TLS*:

- **Protocollo handshake:** applicativo che serve a negoziare fra client e server i meccanismi di sicurezza. Il client contatta un server (ad esempio di posta elettronica). Non posso sapere a priori quali algoritmi di cifratura o quali funzioni hash supporta. Non possiamo pretendere che tutti i nodi siano configurati allo stesso modo. Con questo protocollo è possibile negoziare i protocolli da usare e le chiavi da utilizzare per realizzare autenticazione e riservatezza.
- **Protocollo alert:** protocollo di livello applicativo per gestire i messaggi di errore.
- **Protocollo changecipherspec:** anche questo è un protocollo di livello applicazione e serve per rinegoziare i parametri della connessione.
- **Protocollo record:** protocollo di livello trasporto che riceve dati dal livello applicazione, li trasforma e li incapsula aggiungendo intestazioni di livello 4. Proprio perché i dati sono trasformati a livello trasporto si usa definire SSL/TLS un protocollo di livello trasporto.

10.1.1 TLS e HMAC

SSL/TLS non utilizza firma digitale ma **HMAC**; per questo motivo non è possibile garantire ripudiabilità. Se viene richiesto tale requisito dobbiamo provvedere ad utilizzare SSL/TLS con altri meccanismi.

Perché si è scelto di utilizzare HMAC invece di (ad esempio) MAC con ECB? In generale, quando si effettua una scelta progettuale si valutano sempre tre aspetti:

- **robustezza:** in questo sono uguali;
- **scalabilità:** entrambi usano un segreto simmetrico, quindi entrambi sono confrontabili in termini di scalabilità;
- **efficienza:** HMAC è il più efficiente.

10.2 IPSec

A livello rete il protocollo più usato al momento è **IPv4**, protocollo che però è soggetto ad alcune problematiche:

- **Packet Sniffing**: lettura dei pacchetti in transito;
- **IP spoofing**: falsificazione dell'indirizzo del mittente;
- **Connection hijacking**: inserimento di dati nei pacchetti in transito;
- **Clogging**: generazione di un carico inutile ed oneroso

È per questo motivo che si utilizza **IPSec**, una suite costituita da tre protocolli (che vedremo a breve) in grado di fornire:

- integrità dei datagrammi;
- autenticazione dell'origine dei dati;
- scarto dei pacchetti prodotti replica;
- confidenzialità;
- confidenzialità parziale del traffico.

Più precisamente IPSec può essere definito come **un'architettura per dare sicurezza al livello IP**.

I protocolli che compongono IPSec sono:

- **AH** (*Authentication Header*): per l'autenticazione dei pacchetti;
- **ESP** (*Encapsulating Security Payload*): per confidenzialità ed autenticazione dei pacchetti;
- **IKE** (*Internet Key Exchange*): per la negoziazione dei parametri di sicurezza, l'autenticazione e la distribuzione delle chiavi.

Le funzioni dei protocolli **AH** ed **ESP** (con o senza autenticazione) possono essere riassunte in una tabella:

6CAPITOLO 10. SERVIZI SICURI PER LA COMUNICAZIONE IN RETE

	AH	ESP cifr.	ESP cfr.aut.
Integrità	•		•
Autenticazione	•		•
Rifiuto repliche	•	•	•
Confidenzialità		•	•
Confid. parziale		•	•

Oltre alla suite di protocolli appena vista, IPSec prevede un insieme di **entità** per istanziare i servizi:

- **SPD (Security Policy Database)**: è l'entità che esamina tutto il traffico IP in ingresso e in uscita per decidere quali pacchetti devono usufruire dei servizi di IPSec e per approntare di caso in caso il tipo più adatto. Mette in relazione una porzione del traffico IP ad *SA* specifiche (vedere più avanti) oppure a nessuna *SA* se al traffico IP è consentito aggirare i controlli di IPSec. Un SPD contiene un insieme di politiche (*policy entries*) ciascuna delle quali specifica una porzione del traffico IP e la SA per quella particolare porzione di traffico. Queste porzioni di traffico vengono individuate da un insieme di selettori come indirizzo IP di destinazione, sorgente, user id, livello di sensibilità dei dati, protocollo di livello trasporto, protocollo IPSec usato, porta sorgente e di destinazione ecc... Tramite i selettori si accede all'SPD. Un esempio di politica è: "*tutto il traffico UDP, porta 4015 verso 192.168.1.2 dev'essere scartato!*".
- **SAD (Security Association Database)**: è l'entità che tiene traccia di quale tipo di servizio sia stato espletato, in quale modo e dove siano stati indirizzati i pacchetti in questione. Definisce i parametri associati a ciascuna SA.
- **SA (Security Association)**: struttura che identifica in maniera univoca una connessione unidirezionale tra due interlocutori, specificando il servizio di sicurezza offerto con i relativi parametri (chiavi, algoritmi..). Per avere protezione completa di un canale bidirezionale occorrono due connessioni logiche unidirezionali. Una SA è identificata da tre parametri:
 - **SPI (Security Parameter Index)**;
 - **IP di destinazione**;

– **Protocollo di sicurezza usato (AH o ESP).**

Fra i parametri che troviamo all'interno di una SA vi sono:

- contatore del numero di sequenza;
- finestra di anti-reply;
- informazioni relative a AH;
- informazioni relative a ESP;
- durata dell'associazione;
- modalità operativa di IPSec (tunnel, transport o wildcard);
- path MTU.

Come avviene l'invio di un messaggio con IPSec? Lo vediamo dallo schema in figura 10.1.

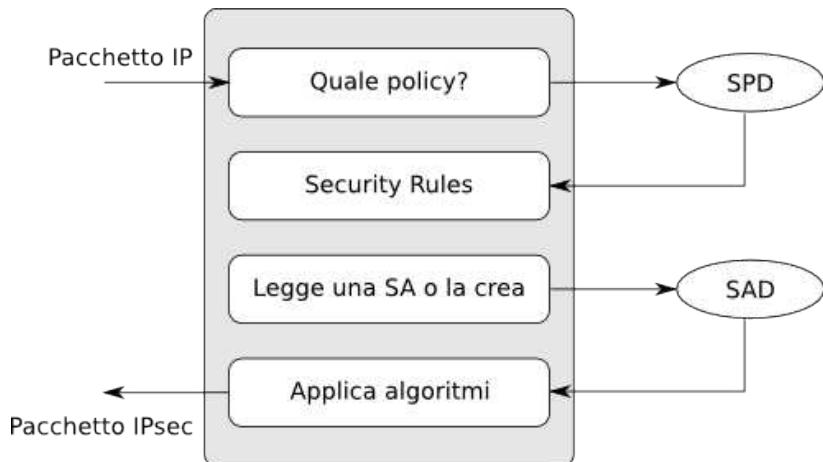


Figure 10.1: Invio di un messaggio con IPSec

Mentre la ricezione di un messaggio segue lo schema in figura 10.2.

10.2.1 Il servizio per l'autenticazione AH

In figura 10.3 possiamo vedere come viene garantita l'autenticazione tramite AH: del pacchetto IP si crea un hash con HMAC (sfruttando dunque un segreto condiviso). Dal momento che si utilizza HMAC sono garantite sia l'autenticazione (visto che per generare l'HMAC si usa un segreto condiviso), sia l'integrità (perché l'hash generato ci permette di vedere se il messaggio è

8CAPITOLO 10. SERVIZI SICURI PER LA COMUNICAZIONE IN RETE

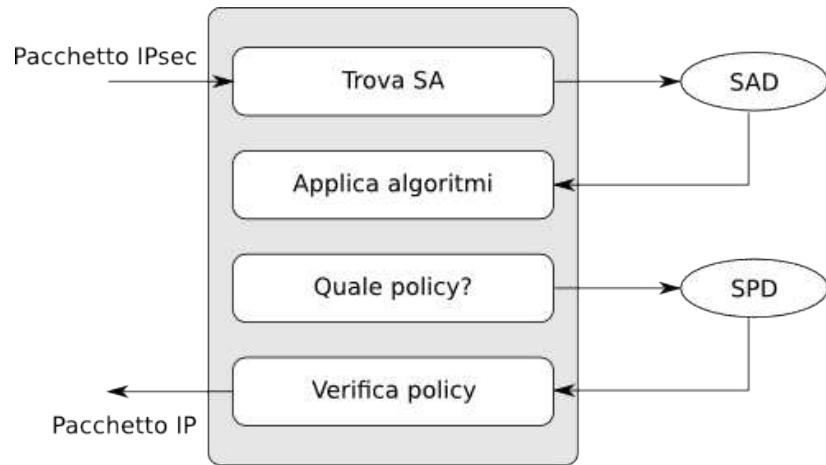


Figure 10.2: Ricezione di un messaggio con IPsec

integro). Dal lato del ricevente si crea allo stesso modo l'HMAC per verificare l'integrità e l'autenticazione del pacchetto IP. L'HMAC può esser fatto con *MD5*, *SHA-1* e via dicendo.

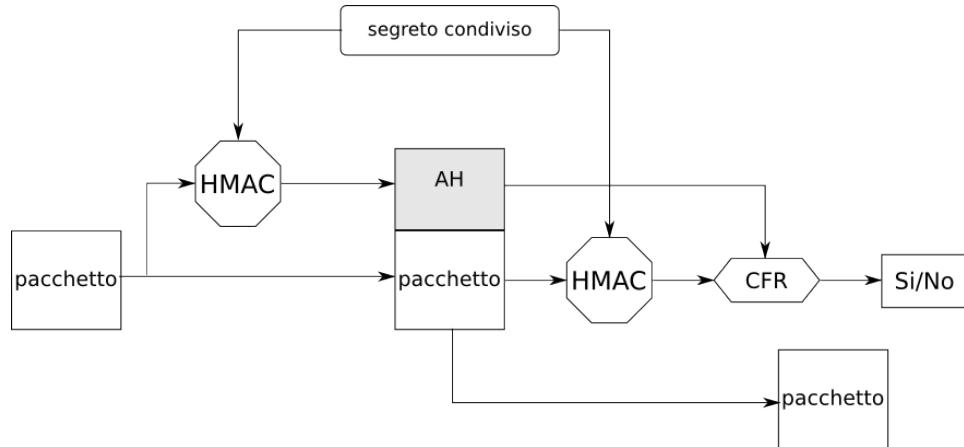


Figure 10.3: Il servizio per l'autenticazione di AH

10.2.2 Il servizio per la riservatezza ESP

In figura 10.4 è possibile osservare come viene garantita la riservatezza tramite ESP. Il pacchetto IP creato viene cifrato tramite una chiave segreta condivisa

ed inviato. Il ricevente lo decifrerà utilizzando la stessa chiave. La cifratura garantisce segretezza, l'uso di una chiave segreta garantisce l'origine. La cifratura può essere realizzata con DES, TDES, IDEA ecc...

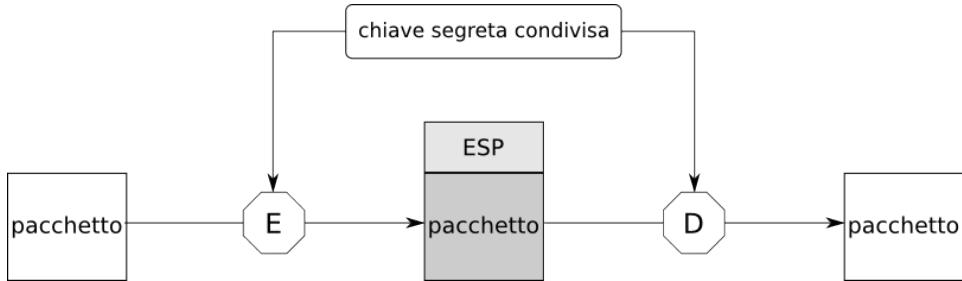


Figure 10.4: Il servizio per la riservatezza di ESP

10.2.3 Le intestazioni di AH e ESP

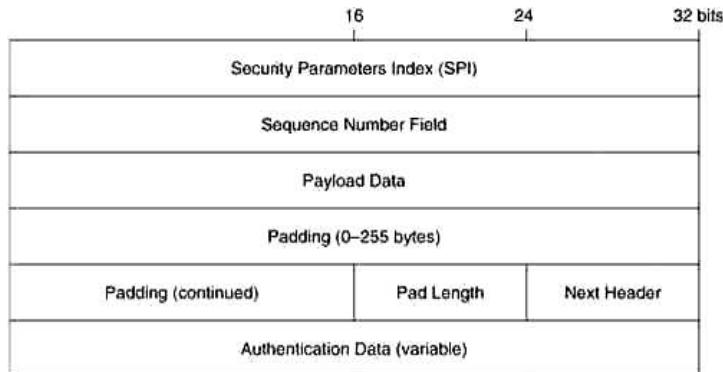
- AH:

8	16	32 bits
Next Header	Payload Length	Reserved
Security Parameters Index (SPI)		
Sequence Number Field		
Authentication Data		

- ESP:

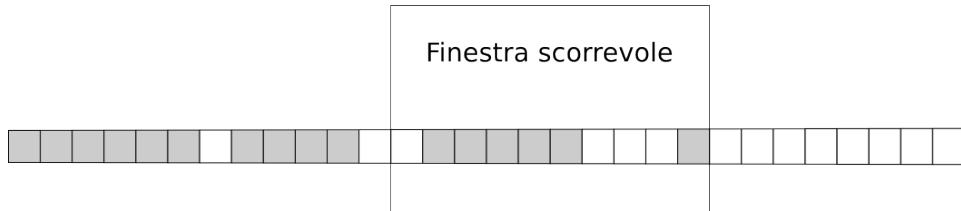
Possiamo notare la presenza di alcuni campi in comune:

- **Sequence Parameter Index:** questo campo di 32 bit, insieme all'indirizzo IP di destinazione e all'identificatore di protocollo, identifica in maniera univoca la SA relativa al datagramma.
- **Sequence Number Field:** espleta il servizio di anti-replay tramite la numerazione progressiva dei datagrammi.
- **Authentication Data:** contiene l'**ICV** (*Integrity Check Value*) che consente di verificare l'integrità del pacchetto in fase di ricezione.



10.2.4 Servizio anti-replay

Come abbiam visto poco fa, il servizio anti-replay viene realizzato tramite il **sequence number field**. Si utilizza un meccanismo a **finestra scorrevole** (di solito di larghezza 64 pacchetti). Ogni pacchetto ha un numero di sequenza. Quando ricevo un pacchetto che si trova a sinistra (fuori dalla finestra) lo scarto, se si trova all'interno della finestra marco la casella corrispondente come occupata, se si trova a destra (fuori dalla finestra), sposto la finestra in modo tale che la casella più a destra sia in corrispondenza con il numero di sequenza estratto.



10.2.5 Trasporto e tunnel

Le due modalità principali consentite da IPsec sono **trasporto** e **tunnel**. In transport mode viene protetto solo il carico del pacchetto IP, cioè i dati provenienti dai protocolli di livello superiore a quello di rete (TCP, UDP, etc); in tunnel mode invece il pacchetto IP originale viene incapsulato in un nuovo pacchetto IP, dopo essere stato elaborato da AH o ESP.

10.2.6 Il protocollo IKE

IKE (*Internet Key Exchange*) è uno dei tre protocolli che compongono IPsec. Si tratta di un meccanismo tramite il quale due interlocutori possono accordarsi sui parametri relativi ad una *SA* (ma non solo visto che consente l'uso di chiavi prestabilite, dette *master keys*). È a sua volta composto da tre protocolli:

- **ISAKMP**: fornisce un framework ed un generico protocollo di negoziazione per stabilire delle SA e chiavi crittografate, ma non stabilisce alcun particolare meccanismo di autenticazione;
- **Oakley**: è una suite di protocolli per l'accordo delle chiavi, che permette a due entità di creare una chiave condivisa; è un'estensione del protocollo *Diffie-Hellman*;
- **Scheme**: usato per la gestione delle chiavi.

Capitolo 11

Firewall

Nel ruolo della protezione della sicurezza un ruolo molto importante è svolto dai **firewall**. Un firewall è un sistema di controllo degli accessi che verifica tutto il traffico che transita attraverso di lui e consente o nega il passaggio del traffico basandosi su delle **security policy**. Il firewall dunque effettua una verifica dei pacchetti in transito (**IP filtering**), ma ha anche il ruolo di mascherare gli indirizzi interni (**NAT**).

Prima di predisporre dei firewall è importante conoscere la topologia della rete in quanto andrebbero installati sui punti di accesso della rete in quanto i firewall possono controllare soltanto il traffico che li attraversa. La rete dev'essere divisa in **zone di sicurezza**, ciò significa dividere le risorse fra le zone in base ai requisiti di sicurezza. La rete acquisisce in questo modo una maggiore scalabilità e stabilità.

Normalmente si tende a inserire fra la rete interna aziendale e la rete esterna una rete detta **zona demilitarizzata (DMZ)**. Nella *DMZ* si inseriscono i server contenenti servizi accessibili dall'esterno, solitamente server la cui eventuale compromissione non causerebbe effetti collaterali.

I firewall mettono a disposizione differenti servizi, oltre a quello di filtraggio dei pacchetti:

- **controllo dei servizi:** stabilire a quali tipi di servizi internet si può accedere sia dall'interno che dall'esterno.
- **controllo della direzione:** stabilire la direzione secondo cui particolari richieste di servizio possono essere avviate e inoltrate.
- **controllo utente:** regolare l'accesso ad un servizio sulla base dell'utente che ha effettuato la richiesta.

- **controllo del comportamento:** controllo su come sono utilizzati certi servizi (ad esempio filtraggio della posta elettronica per eliminare lo spam).

Ormai sappiamo che il firewall legge ed applica le regole configurate, ma bisogna dire che esistono due politiche mutuamente esclusive che possono essere applicate nel seguire le regole:

- **tutto ciò che non è espressamente concesso è vietato:** il firewall blocca tutto il traffico per il quale non esistono regole configurate. Ogni servizio deve essere abilitato caso per caso. Chiaramente questo meccanismo garantisce una maggior sicurezza, ma è più difficile da gestire;
- **tutto ciò che non è espressamente vietato è concesso:** è il caso opposto al precedente. Tutto il traffico per il quale non sono state previste delle regole segue il suo normale andamento. Ciò può rivelarsi pericoloso e rendere sicuro un sistema diventa davvero complesso.

Dal momento che il firewall si occupa di garantire la sicurezza all'interno della rete, dev'essere l'host più protetto della rete. È bene che sia una macchina dedicata, dove non ci siano altri servizi a utilizzare delle risorse che possono rivelarsi preziose per il firewall o che possono rappresentare punti d'attacco per l'intrusore.

11.1 Tipologie di firewall

I firewall vengono suddivisi in due categorie:

- **packet filtering:** viene installato a monte della rete protetta ed ha il compito di bloccare o inoltrare i pacchetti IP secondo regole definite a priori. È quindi un applicativo di livello rete. Un packet filter (*screening router*) scarta o inoltra un pacchetto IP, da e verso la rete interna, sulla base di un insieme di regole di filtraggio che operano su campi dell'intestazione IP e di trasporto. Fra i campi analizzati vi sono l'indirizzo IP sorgente e destinazione, il protocollo di trasporto, numero di porta sorgente e destinazione, i flag *SYN* e *ACK* nell'header TCP.
- **circuit gateway:** analizza e filtra il traffico a livello trasporto. I circuit gateway effettuano l'autorizzazione, il logging ed il caching delle connessioni.

- **application gateway:** analizza e filtra il traffico a livello applicazione sfruttando la conoscenza del particolare servizio. Gli application gateway non sono trasparenti, richiedono carico addizionale e soprattutto per ogni servizio basato su TCP ne è necessario uno. Dalla loro hanno l'autenticazione del client e del server, il filtraggio specifico del servizio per tutto il traffico, il mascheramento della rete ed il logging. Esempi di application gateway sono i proxy.

Definiamo **bastion host** un host critico per la sicurezza che costituisce la piattaforma per i gateway a livello applicazione e di circuito. I bastion host montano un sistema operativo sicuro e soltanto i proxy necessari. I suoi servizi sono destinati ad un sottoinsieme degli host nella rete, implementano forme di autenticazione aggiuntive e specifiche e supportano logging & auditing.

Un particolare tipo di filtraggio è lo **stateful packet filter** con il quale si considera il traffico come uno scambio di pacchetti IP che costituisce una **sessione di conversazione**. Tramite lo stateful packet filter si generano delle **regole dinamiche**. Ogni nuovo pacchetto che soddisfa una regola dinamica provoca la creazione di una nuova regola dinamica per il successivo pacchetto. Nel momento in cui non vengano reperite regole dinamiche adatte al pacchetto ricevuto, si applicano regole statiche. Lo stateful filtering permette di concentrarsi sul blocco o meno di intere sessioni.

11.2 Disporre i firewall

Vediamo adesso alcuni esempi su come disporre i firewall all'interno di una rete.

- **Firewall Single-Homed:** in questa disposizione, il firewall è composto da un *packet filter* posto al confine fra la rete esterna e quella interna e da un *bastion host* posto nella rete interna. Il *packet filter* blocca tutto il traffico proveniente dall'esterno che non sia diretto al *bastion host* o ad un altro server interno alla rete. I pacchetti subiscono dunque un'ulteriore analisi una volta arrivati al *bastion host*. Questo sistema ha però un punto debole, e cioè nel momento in cui viene compromesso il *packet filter*, il traffico esterno può raggiungere la rete interna ed evitare il *bastion host*. Il *bastion host* ha una sola interfaccia di rete.
- **Firewall Dual-Homed:** il *dual-homed bastion host* previene i problemi causati dalla compromissione del *packet filter* in quanto un pac-

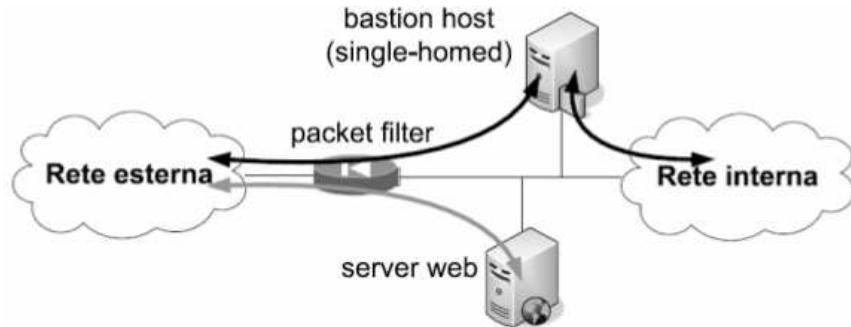


Figure 11.1: Firewall Single Homed

chetto deve fisicamente attraversare il *bastion host* (che ha due interfacce di rete) e questo fa aumentare il livello della sicurezza.

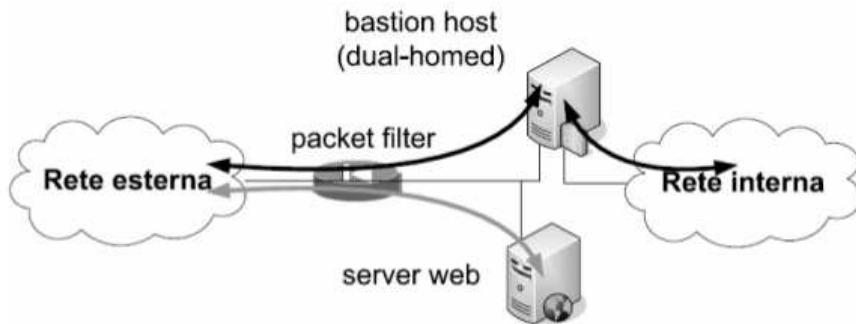


Figure 11.2: Firewall Dual Homed

- **Screened Subnet Firewall:** questa è la configurazione più sicura in quanto crea una sottorete isolata sulla quale abbiamo soltanto il *bastion host* ed i server esposti. Prevede tre livelli di difesa in quanto i pacchetti malevoli devono prima di tutto attraversare il primo *packet filter*, poi essere approvati dal *bastion host*, in seguito riuscire ad oltre passare anche il secondo *packet filter*. Dall'esterno soltanto la sottorete isolata è visibile ed inoltre gli host sulla sottorete interna non vedono al di là della sottorete protetta dunque non possono stabilire connessioni dirette a Internet.

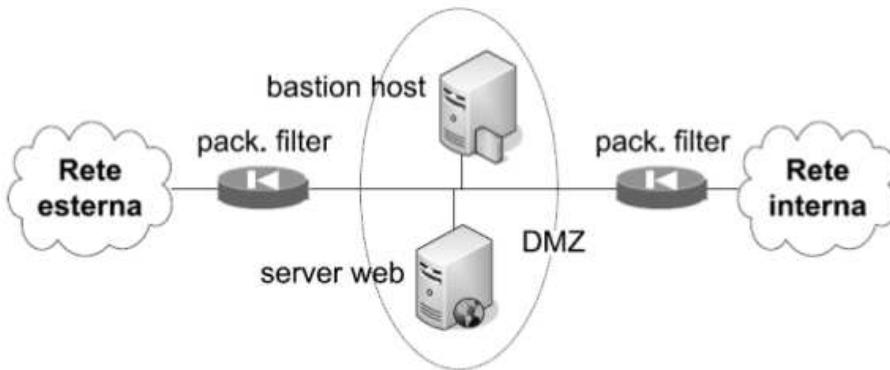


Figure 11.3: Screened Subnet Firewall

In molti casi si vuole accedere dall'esterno a dei servizi messi a disposizione dall'azienda (server web, ftp, posta) o si vuole accedere alla rete aziendale da casa. Cosa si fa in questi casi? Si mette su un'**architettura di rete a due livelli** o si sfruttano le **VPN** (*Virtual Private Network*).

- l'**architettura a due livelli** prevede la creazione di una rete *semipubblica* fra Internet e la intranet, detta **zona demilitarizzata (DMZ)**. L'idea di base è che i servizi accessibili dall'esterno siano posizionati nella *DMZ* e che soltanto a questi si possa accedere dall'esterno. La rete interna rimane nascosta. La *DMZ* è una zona pericolosissima in quanto molto esposta, dunque ogni connessione proveniente dalla *DMZ* la si dovrà trattare come proveniente dall'esterno. Si possono usare due firewall *single homed* oppure un singolo firewall *a tre vie*.
- la **virtual private network** è un ponte crittografico su rete pubblica che fa fronte ad alcune esigenze come comunicare dall'esterno con la rete aziendale, collegare tra loro delle sedi di un'azienda, garantire confidenzialità ed integrità dei dati trasmessi su rete pubblica non sicura.

Capitolo 12

Intrusion Detection System

I meccanismi per la gestione degli attacchi si dividono fra:

- meccanismi di **prevenzione**;
- meccanismi di **rilevazione**;
- meccanismi di **tolleranza** (*recovery*).

In questo capitolo studieremo i meccanismi di **Intrusion Detection**, tecnica che può essere attiva o passiva, automatica o manuale. Gli **Intrusion Detection System** sono meccanismi automatici (ma non solo) per la rilevazione delle intrusioni. Vedremo che gli *IDS* possono essere classificati fra diversi tipi. Non è possibile usare solo approcci automatici o solo approcci manuali: l'*intrusion detection* necessita di entrambi in quanto gli approcci automatici potrebbero non coprire tutti i possibili attacchi, mentre gli approcci solo manuali potrebbero dover analizzare una mole di dati troppo grande.

12.1 Obiettivi di un IDS

L'obiettivo di un *IDS*, oltre chiaramente a rilevare le intrusioni, è quello di minimizzare il numero di falsi negativi e quello di falsi positivi. Un **falso positivo** è una situazione in cui l'*IDS* notifica all'amministratore un evento eccezionale, ma in realtà non si tratta di un'intrusione. Un **falso negativo** è il caso in cui un'intrusione è in corso, ma il sistema non riesce a rilevarla. Il primo è pericoloso in quanto dopo un elevato numero di falsi positivi l'amministratore smette di ascoltare l'*IDS*, il secondo è pericoloso

in quanto il sistema può essere compromesso. È importante fare attenzione perché minimizzare i falsi positivi può portare a ignorare erroneamente i falsi negativi. Un *IDS* dev'essere anche in grado di proteggere sè stesso da manomissioni (**subversion**).

12.2 Caratteristiche di un buon IDS

Prima di introdurre i componenti e la classificazione fra gli IDS, citiamo le principali caratteristiche di un buon IDS:

- **non** dev'essere una **scatola nera**;
- dev'essere **fault tolerant**, cioè dev'essere in grado di resistere ai guasti senza che sia necessario ristabilire la sua *knowledge-base*;
- dev'essere in grado di monitorare se stesso per **resistere alle manomissioni**;
- deve imporre un **overhead minimo** sul sistema target;
- dev'essere **customizzabile** in base al sistema target;
- dev'essere in grado di **evolversi** per adattarsi ad un sistema target in evoluzione.

12.3 Componenti di un IDS

Un *IDS* si compone delle seguenti parti

- **Sensori**: raccolgono le informazioni come log, pacchetti, system call...
- **Meccanismi di analisi**: determinano se è avvenuta un'intrusione.
- **Interfaccia utente**: permette di interagire con l'*IDS*.
- **Honeypot**: è un componente addizionale. Si tratta di una trappola, un sistema intenzionalmente debole che offre dei *finti* servizi allo scopo di attirare gli attaccanti e rilevare il loro comportamento.

12.4 Classificazione degli IDS

La classificazione degli *IDS* può avvenire sulla base di alcuni parametri:

- **Architettura utilizzata**
- **Temporizzazione**
- **Sorgente delle informazioni**
- **Tipo di analisi svolta**
- **Tipo di risposta**

12.4.1 Classificazione sulla base dell'architettura

Iniziamo definendo come **host** il dispositivo su cui viene eseguito l'*IDS*, come **target** il dispositivo oggetto dell'analisi. Prendendo come parametro l'architettura con cui l'*IDS* è realizzato, possiamo distinguere fra **co-locazione Host-Target** e **separazione Host-Target**. Nel primo caso *host* e *target* coincidono, ma questo pur essendo un sistema molto comune, aumenta il rischio di *subversion*. Nel secondo caso *host* e *target* sono separati e perciò per compromettere il sistema senza essere rilevati bisogna prima compromettere l'*IDS* e compiere dunque due intrusioni.

Nel caso di separazione fra *host* e *target* possiamo adoperare un'**architettura centralizzata** in cui tutte le operazioni di monitoraggio vengono svolte da una posizione centralizzata, oppure su un'**architettura distribuita** o **parzialmente distribuita**.

12.4.2 Classificazione in base alla temporizzazione

Sulla base della **temporizzazione** possiamo parlare di sistemi ad **analisi periodica** (*batch mode*) e **analisi continua** (*real-time*). I primi non permettono risposte attive e funzionano analizzando i log, i secondi si basano su un flusso costante di informazioni e ciò permette risposta attiva ed in tempo reale.

12.4.3 Classificazione in base alle sorgenti di dati

I sistemi di *intrusion detection* possono essere classificati anche sulla base delle sorgenti di dati da analizzare. Possiamo avere sistemi:

- **Host-based:** effettua l'*audit* dei dati provenienti da un singolo host.

- **Vantaggi:** possono rilevare attacchi interni ad un host (accessi ai file, cambiamenti negli account, cambiamenti di politiche...), possono verificare se un attacco ha avuto successo (tramite i log), possono operare in ambienti in cui il traffico di rete è criptato, possono rilevare attacchi che coinvolgono l'integrità del software.
- **Svantaggi:** sono difficili da gestire in quanto richiedono configurazione ad-hoc su ogni host, sono parzialmente basati su co-locazione host-target dunque più soggetti a subversion, sono meno performanti.
- **Network-based:** effettua l'*audit* del traffico di rete;
 - **Vantaggi:** sono più economici perché richiedono un numero di componenti minore, sono in grado di rilevare attacchi che un sistema host-based non può rilevare (attacchi IP-based o DOS ad esempio), è più difficile per l'attaccante rimuovere le tracce, possono operare in real time, sono indipendenti dal sistema operativo.
 - **Svantaggi:** possono avere difficoltà nel processare i dati di una rete molto trafficata, molti dei vantaggi non si applicano su reti basate su switch, non possono analizzare pacchetti crittografati, non possono riconoscere se un attacco è riuscito o fallito.
- **Application-based:** specializzato per applicazioni specifiche;
 - **Vantaggi:** permettono di tracciare attività consentite o vietate all'utente e possono operare in ambienti in cui il traffico è criptato.
 - **Svantaggi:** sono più vulnerabili a subversion perché i loro log sono meno protetti di quelli del sistema operativo e sono spesso vulnerabili ad attacchi di tipo trojan horse.
- **Approccio distribuito:** combina i tre precedenti approcci per scoprire intrusioni più complesse.
 - **Vantaggi:** prende il meglio dai metodi precedenti.
 - **Svantaggi:** architettura più complessa.

In realtà si può seguire una gerarchia secondo la quale un *Application IDS* passa i dati ad un *Host IDS*, questo li passa ad un *Network IDS* il quale li fornisce ad un *Distributed IDS*.

12.4.4 Classificazione sul modello di rilevazione delle intrusioni

Vediamo secondo quali modelli possono rilevare le intrusioni.

- **Anomaly Detection Model:** le intrusioni sono rilevate cercando attività che si differenziano dal normale comportamento di un utente.
- **Misuse Detection Model:** le intrusioni sono rilevate cercando di scoprire attività che corrispondono a tecniche note di intrusione.

L'**Anomaly Detection Model** si basa sul fatto che infrangere la sicurezza di un sistema comporta spesso un uso anormale del sistema. Si crea dunque un profilo (con un'analisi sul lungo periodo) per ogni utente, macchina e rete costituito da **metriche** ovvero variabili che rappresentano particolari misure quantitative. Esempi di tali variabili sono il carico medio della CPU, il numero di processi per utente, il numero di connessioni di rete. Le tecniche utilizzate in questo modello possono essere:

- **Threshold:** viene stabilita una soglia per ognuna delle metriche.
- **Rule-based:** il comportamento accettabile non viene stabilito tramite soglia, ma tramite regole complesse.
- **Misure statistiche:** analisi della distribuzione statistica delle richieste. È un caso particolare del precedente.

Pro e contro: il modello di *Anomaly Detection* è in grado di rilevare i sintomi di un attacco senza conoscerne i dettagli, semplicemente rilevando delle anomalie nel comportamento abituale. Possono anche rilevare nuovi attacchi per i *misuse detection*. Di contro tali sistemi possono produrre un alto numero di falsi positivi e richiedere una grande quantità di training da parte dell'amministratore del sistema. Attualmente vengono usati per lo più in ambito di ricerca, in quanto le tecniche di anomaly detection sono ancora sperimentali.

I modelli **Misuse Detection** operano con un **dizionario di attacchi** e ciò che fanno è semplicemente *pattern matching*. Esistono due approcci:

- **Signature-based:** ogni attacco è una sequenza di eventi ed ogni attacco è caratterizzato dalla sua *signature*.
- **State-based:** ogni attacco è una sequenza di azioni.

Pro e contro: i modelli *Misuse Detection* sono molto efficaci nel rilevare attacchi senza generare falsi positivi e richiedono poca conoscenza all'amministratore, tuttavia sono in grado di rilevare attacchi già noti pertanto devono essere aggiornati frequentemente onde evitare falsi negativi. Data l'assenza di falsi positivi vengono utilizzati in sistemi commerciali (a volte integrando tecniche di anomaly detection).

Analizziamo il problema della **Knowledge-base**, ovvero della base di conoscenza necessaria al funzionamento dei due modelli. Nel caso dei modelli **anomaly detection** il sistema dev'essere istruito sul comportamento normale, è compito dell'**amministratore** di sistema e può essere automatizzato. Nei modelli **misuse detection**, il sistema possiede una base di dati contenente i possibili attacchi e viene istruito in questo modo sui comportamenti anormali. È compito del **produttore** dell'IDS.

12.5 Notifiche agli utenti

Gli utenti di un sistema devono essere informati della presenza di controlli e del loro scopo, nonché delle conseguenze di comportamenti non autorizzati. Un modo di notificare tutto ciò è ad esempio un messaggio a video.

12.6 Integrità del software

Uno degli aspetti fondamentali di un IDS è l'integrità sia dell'IDS, sia del sistema operativo. Esistono cinque modi per garantire l'integrità:

- **spostamento fisico del disco:** si analizza il disco sospettato su un altro sistema;
- **collegamento di un disco verificato:** si collega al sistema sospetto un disco verificato e protetto in scrittura contenente il software necessario a verificare l'integrità;
- **creazione di un'immagine del disco sospettato:** si crea l'immagine e la si verifica altrove;
- **utilizzazione di strumenti esterni:** si utilizzano software provenienti ad esempio da cd-rom;
- **verifica del software installato:** si confronta il software installato con una copia di riferimento.

Capitolo 13

Architettura di sicurezza di Java

13.1 Il class loader di Java

Il class loader di Java ha tre proprietà fondamentali:

- **lazy loading:** le classi vengono caricate *on demand* e possibilmente all'ultimo momento. Possiamo vedere un esempio nel seguente blocco di codice:

```
1 Prova p = new Prova();  
2  
3 Attrib1 a1;  
4 Attrib2 a2=new Attrib2();  
5  
6 initAttrib1() { a1=new Attrib1(); }
```

La classe *Attrib2* viene caricata alla riga 4, *Attrib1* soltanto alla riga 6;

- **programmabilità:** può essere modificato ed esteso;
- **creazione di namespace multipli:** è possibile creare namespace multipli e mantenere isolati gli ambienti che ne fanno uso.

Il class loader gioca un ruolo fondamentale in quanto (come abbiamo appena detto) crea spazi di nomi separati e perché si coordina con il **Security Manager** che implementa le politiche di sicurezza (associando le classi a domini di protezione).

Come funziona il class loader di Java? Il suo comportamento è molto semplice:

1. **loading:** è il caricamento della classe ed in questa fase si ottengono informazioni sui suoi dati;
2. **super-class loading:** caricamento delle superclassi che avviene dopo aver analizzato la definizione della classe;
3. **risoluzione della classe:** in questa fase avviene la **verification**, cioè viene verificata la struttura della classe e si ispeziona il suo contenuto per assicurarsi che non effettui operazioni proibite. Può anche essere ritardata in quanto non tutte le classi devono essere immediatamente caricate;
4. **restituzione dell'istanza:** viene inizializzata la classe e restituita l'istanza.

La classe che implementa il class loader in Java è **ClassLoader**. Alcuni dei metodi che la compongono sono *loadClass*, *findLoadedClass*, *findSystemClass*, *findClass*, *resolveClass*, *defineClass*.

Ridefinire il class loader di default per creare un proprio class loader è semplice. È sufficiente ereditare dalla classe *ClassLoader* e ridefinire i metodi che ci interessano:

```
1 class MyClassLoader extends ClassLoader {  
2     ...  
3 }
```

In realtà in Java esistono differenti class loader:

- **class loader interno:** viene utilizzato solo per caricare classi di Java;
- **secure class loader:** permette di implementare politiche di sicurezza più granulari;
- **applet class loader:** è implementato (in maniera anche diversa) da ogni browser;
- **URL class loader:** permette di caricare classi da un insieme di URL distinti.



La creazione di class loader personalizzati è permessa a patto che si estendano i tipi predefiniti. Tutti i class loader esistenti derivano in qualche modo da quello che chiamiamo **Primordial Class Loader**:

Il **Primordial Class Loader** è il class loader che fa il bootstrap del processo di caricamento ed è generalmente scritto in linguaggio nativo. Il class loader che viene utilizzato per caricare/definire una classe viene detto **defining class loader**. Il class loader che *avvia* il caricamento di una classe può anche essere diverso da quello che effettivamente la carica ed è detto **initiating class loader**.

A tempo di compilazione il tipo di una classe corrisponde a un nome. **A runtime il tipo effettivo della classe è determinato dalla coppia nome della classe e defining class loader: $\langle C, L \rangle$** . Segue che due tipi di classe nell'ambiente Java sono uguali solo se sono uguali sia i tipi delle classi, sia i *defining class loader*.

Come si fa a caricare una classe o a delegarne il caricamento? Se su un'istanza del class loader si richiama il metodo **defineClass** allora si sta definendo la classe, se si richiama **loadClass** si sta delegando al class loader di sistema il caricamento della classe passata come argomento. Ogni classe C è permanentemente associata al proprio defining class loader ed è lui che carica ogni classe definita all'interno di C .

La Java Virtual Machine mantiene spazi di nomi separati fra user defined class loader e class loader gestito dal compilatore e ne mantiene la consistenza.

Il metodo *loadClass* può ritornare tipi differenti per uno stesso nome di classe in istanti di tempo diversi.

Per mantenere la *type safety* la JVM deve essere in grado di ottenere lo stesso tipo di classe per un dato nome di classe e class loader.

Approccio: la JVM non si fida a priori che uno *user-defined loader* ritorni sempre lo stesso tipo perciò mantiene una cache delle classi caricate. Anziché richiamare nuovamente il metodo *loadClass* per una classe già caricata, si esegue il metodo *findLoadedClass* che la recupera dalla cache.

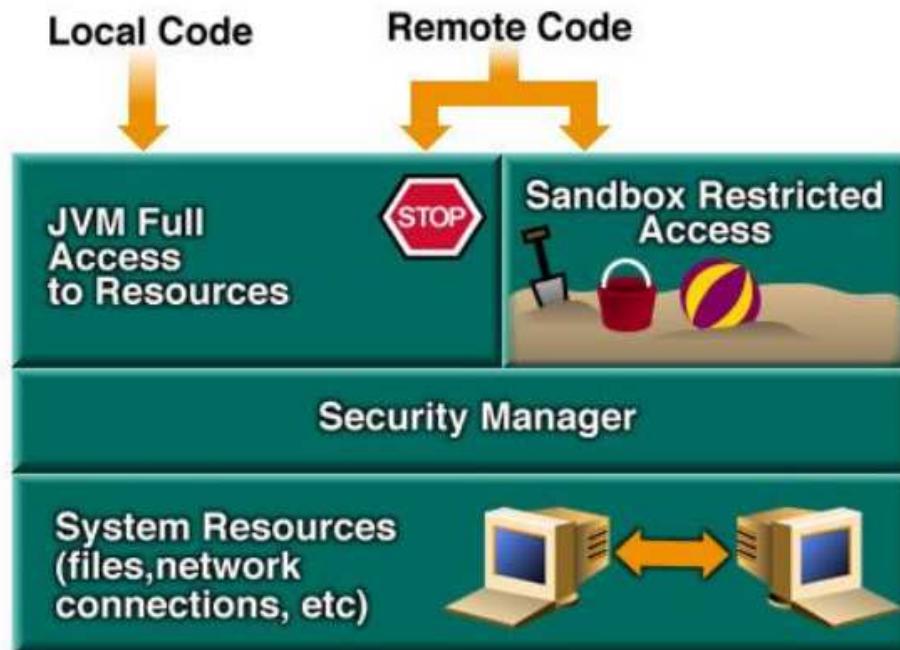


Figure 13.1: JDK 1.0

13.2 Modelli di sicurezza

13.2.1 JDK 1.0

Le applet non possono accedere in nessun caso alle risorse di rete o del sistema, pertanto le applicazioni sono sicure di default. Le applet operano infatti in una **sandbox** che protegge l'accesso a tutte le risorse del sistema. I programmati **possono scrivere** un *Security Manager* per accedere alla *sandbox*.

Il modello è però troppo restrittivo in quanto i programmati **devono scrivere** un proprio *Security Manager* per accedere alla sandbox e in quanto per ogni nuova politica che si vuole implementare bisogna rilasciare una nuova versione del *Security Manager*.

13.2.2 JDK 1.1

In JDK 1.1 è stato introdotto il concetto di firma del codice che assicura **autenticazione** ed **integrità**. Questo però vale solo per il codice remoto;

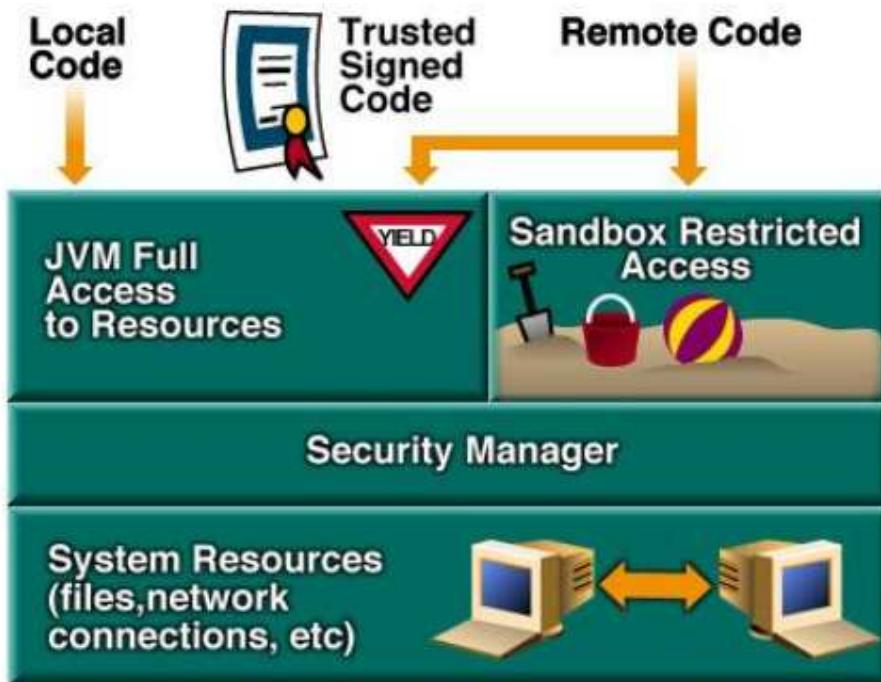


Figure 13.2: JDK 1.1

quello locale non è sottoposto ad alcun tipo di controllo. Tutte le classi contenute nel *classpath* sono considerate fidate.

13.2.3 JDK 1.2

La JDK 1.2 si è posta come obiettivi l'estensione dei controlli a tutto: sia alle applet sia alle applicazioni locali o remote. La nuova architettura di sicurezza è configurabile in maniera granulare e semplificata ed è possibile specificare **politiche d'accesso** in maniera semplice. Per fare ciò sono stati introdotti i concetti di **permessi d'accesso** e di **domini di protezione**.

I permessi di sicurezza sono caratterizzati da un **tipo** (permessi su file, su socket ecc..), da un **target name** (ovvero l'oggetto cui sono riferiti i permessi), dalle **azioni** consentite. I **domini di protezione** descrivono gli oggetti accessibili da una data entità computazionale e con quali permessi. Ogni classe appartiene a un dominio: il runtime mantiene il mapping fra codice della classe e dominio associato.

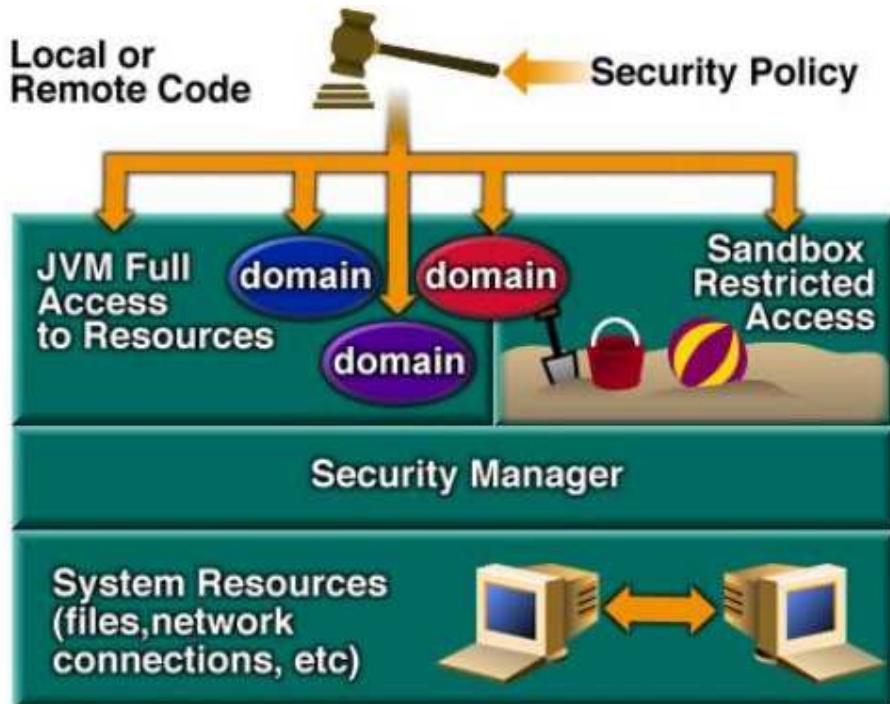


Figure 13.3: JDK 1.2

La JDK 1.2 permette di specificare una vera e propria **politica di sicurezza**. La politica di sicurezza è definita in un file di testo da cui si ricava poi la **matrice degli accessi**. Le asserzioni di sicurezza vengono definite tramite un codice come il seguente:

```

1 grant signedBy "*", codeBase "http://java.sun.com/
   people/gong/" {
2   permission java.io.FilePermission "read, write", "tmp
   /*";
3 }
```

mentre la tabella assume un aspetto come quello che segue:

Codice	Permessi
Li Gong applet	read, write /tmp and /home/gong

Tali asserzioni vengono mantenute in un oggetto di tipo **Policy**. Questo

modello è detto **code centric**. La politica di sicurezza mantiene informazioni sulla **provenienza del codice** (*CodeSource*) e sui **permessi** ad esso attribuiti (tramite la classe Policy si può accedere a metodi come *getPermissions*). Ad esso accede il *Security Manager* per controllare il rispetto dei permessi d'accesso.

In JDK 1.2 viene inoltre implementato un **livello di indirezione** che permette di associare alle classi caricate dei **domini di protezione differenti** sulla base delle politiche definite. Un dominio di protezione descrive gli oggetti accessibili da una data entità computazionale e con quali permessi. Ogni classe appartiene a un dominio: a runtime si mantiene il mapping fra codice della classe e dominio associato. I domini di protezione contengono:

- un oggetto di tipo **Codesource** che descrive l'origine del codice e gli eventuali certificati;
- l'elenco delle entità che eseguono quel codice in un dato istante;
- un riferimento al class loader che ha definito la classe;
- l'elenco dei permessi garantiti staticamente all'atto del caricamento della classe ed i permessi assegnati dinamicamente.

Java fornisce 11 permessi standard (ognuno implementato come classe), ma è possibile definirne di nuovi aggregando più oggetti di classe *Permission* in una *PermissionClass*.

I permessi sono organizzati gerarchicamente come segue:

Confrontiamo come avviene il controllo dell'accesso in JDK 1.1 e 1.2. In JDK 1.1:

```

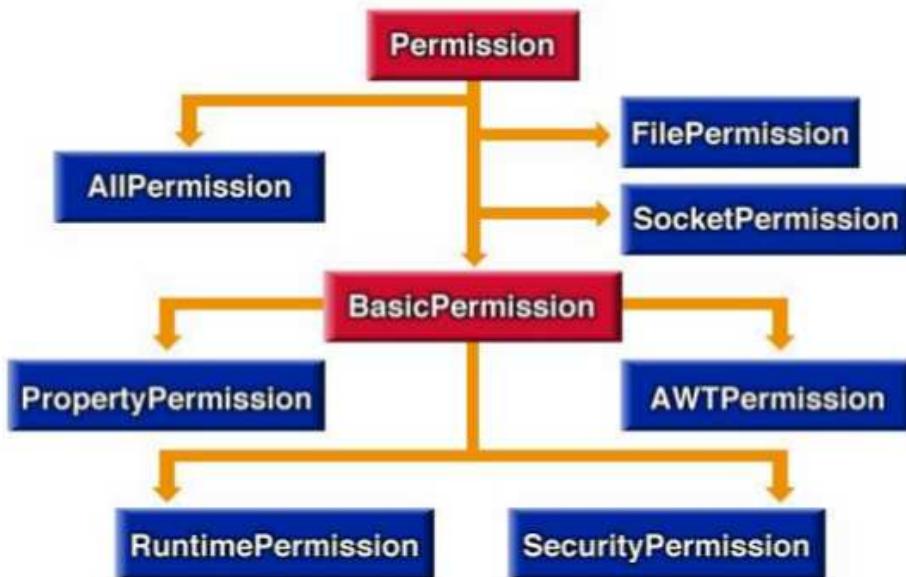
1 SecurityManager security=System .getSecurityManager () ;
2
3 if (security !=null) {
4     Security .checkRead ("path / file ") ;
5 }
```

mentre in JDK 1.2:

```

1 FilePermission perm = new
2 FilePermission ("path / file ", "read ") ;
3 AccessController .checkPermission (perm) ;
```

Come possiamo notare, in questa seconda versione il controllo dell'accesso è allo stesso tempo *generico* e *tipizzato*.



Un componente importante è l'**Access Controller**. Questo controlla se tutti i domini delle chiamate all'interno di un thread hanno il permesso richiesto. Questo può esser fatto secondo una strategia **eager evaluation** o **lazy evaluation**.

Ai domini devono essere garantiti dei diritti tali da non violare il principio del minimo privilegio garantito.

L'algoritmo di controllo dell'accesso può essere facilmente implementato come segue:

```

1 void checkPermission(Permission p) {
2     foreach (caller) {
3         if (the caller doesn't have permission)
4             throw new AccessControlException(p);
5         if (caller is marked as privileged)
6             return;
7     }
8     // Access Granted
9     return;
10}
  
```

La chiamata del metodo `checkPermission` provoca lo snapshot dello stack

del thread corrente e per ogni chiamata nello stack si chiama il metodo *implies* che provoca la verifica attuale dei permessi associati a quel dominio di protezione.

Se la valutazione della politica è dinamica si ha la valutazione dinamica dei permessi associati ad ogni dominio di protezione; in particolare si invoca il metodo *implies* dell'oggetto *Policy* installato sul sistema che per quel dominio di protezione ritrova la collezioni di permessi associati e su ognuno di essi invoca il metodo *implies*.

13.2.4 In sintesi...

Riassumendo possiamo dire che il modello di sicurezza di JDK 1.2 ha introdotto due vantaggi che sono la separazione delle politiche dai meccanismi e l'estensibilità e scalabilità dei controlli di sicurezza. Di contro però è difficile controllare l'uso di risorse.

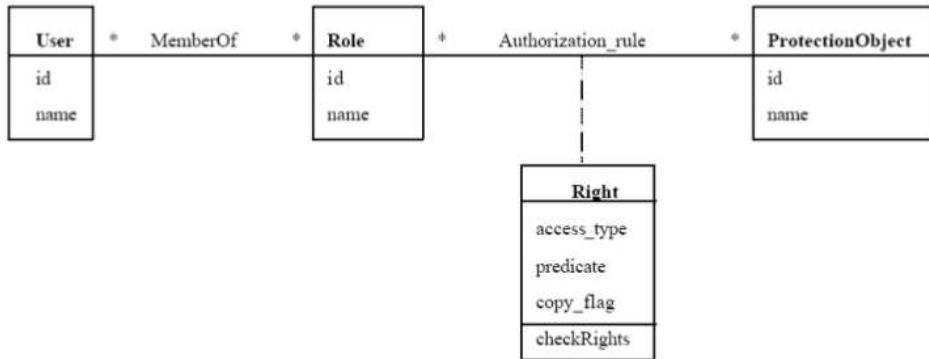
Il comportamento di JRE è specificato dalla politica adottata che viene specificata esternamente da file ASCII, internamente dalla classe Policy ricavata dalla matrice degli accessi. Si può consultare l'oggetto Policy per sapere quali privilegi sono garantiti al codice:

- 1 Permissions permissions = Policy . getPolicy () .
getPermissions (codesource) ;

Capitolo 14

RBAC

I sistemi di tipo **Role Based Access Control (RBAC)** assegnano i privilegi non agli utenti, ma alla funzione che questi possono svolgere nel contesto di una certa organizzazione.



RBAC consente di rispettare facilmente i principi di sicurezza:

- **minimo privilegio:** un utente non deve avere mai più privilegi di quelli che sono necessari;
- **separazione dei compiti:** è opportuno fare in modo che il controllato non sia anche il controllore, dunque si richiedono ruoli mutuamente esclusivi in particolari situazioni critiche.

Inoltre RBAC permette di avere **astrazione dei dati** in quanto ad esempio invece dei tipici permessi *read*, *write* ed *execute* si può pensare ad un

modello basato su un concetto astratto che fa uso ad esempio di *crediti* e *debiti*.

In RBAC un **ruolo** è il costrutto semantico intorno al quale si formulano le politiche di controllo dell'accesso. Al concetto di ruolo possono essere attribuiti due diversi significati:

- rappresenta la **competenza** nel compiere una specifica attività (ad esempio *fisico* o *farmacista*);
- incorpora sia l'**autorità** che la **responsabilità**, ad esempio *responsabile del progetto*.

I sistemi RBAC forniscono **grande flessibilità** agli amministratori di sistema in quanto i ruoli sono quasi fissi (variano raramente) e stabiliti una volta i permessi d'accesso per i ruoli, ciò che resta da fare all'amministratore è assegnare gli utenti ai ruoli. Segue che la politica RBAC è più vantaggiosa rispetto alle politiche DAC e MAC.

RBAC è stato standardizzato da ANSI che ha proposto quattro modelli in modo incrementale:

- **Core RBAC;**
- **Hierarchical RBAC;**
- **RBAC con vincoli SSD;**
- **RBAC con vincoli DSD.**

14.1 Core RBAC

In questo modello vengono definiti solo gli elementi essenziali senza i quali non è possibile implementare un controllo d'accesso.

I concetti essenziali sono:

- **Utente:** un essere umano o un software;
- **Ruolo:** è una *funzione lavorativa* all'interno di un sistema; descrive le responsabilità conferite al membro del ruolo. Se esiste una relazione fra un utente ed un ruolo non vuol dire che l'utente appartiene a quel ruolo sempre, ma solo che gli è permesso farne parte;
- **Permesso:** è l'approvazione di un particolare diritto di accesso ad uno o più oggetti (risorse) del sistema;

- **Sessione:** connettendosi al sistema l'utente stabilisce una sessione nella quale può attivare (anche solo) un sottoinsieme dei ruoli che gli appartengono. Ogni sessione tiene traccia dell'utente e dei ruoli che può attivare. Un utente assume un ruolo attivando la corrispondente sessione (ad esempio l'utente attiva la sessione *amministratore*) e abbandonerà il ruolo disattivandola. Un utente può creare più sessioni all'interno delle quali può attivare uno o più ruoli. I permessi messi a disposizione dell'utente sono dati dall'unione dei permessi associati ai ruoli attivi nella sessione.

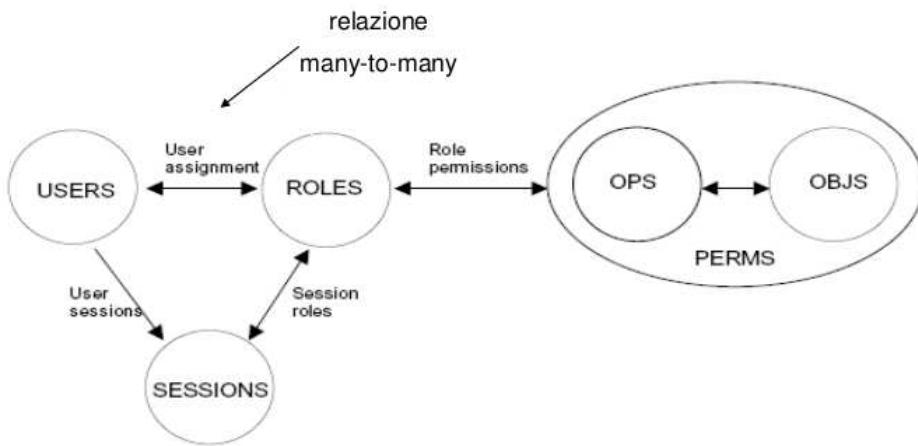


Figure 14.1: Core RBAC

Implementare un modello Core RBAC vuol dire fornire un sistema con cui è possibile interagire mediante:

- **funzioni di amministrazione:** creare e rimuovere utenti, ruoli, risorse, permessi;
- **funzioni di supporto:** creare sessioni, aggiungere ruoli ad una sessione o rimuovere ruoli attivi;
- **funzioni di monitoraggio:** controllare il sistema.

14.2 Hierarchical RBAC

Una gerarchia è un modo naturale di strutturare i ruoli all'interno di un'organizzazione che rispecchia l'effettiva responsabilità e autorità di ognuno.

Ad una gerarchia corrisponde solitamente un'effettiva ereditarietà dei permessi, cioè salendo di livello, i vari ruoli possiedono tutti i permessi dei ruoli sottostanti oltre ai propri. Nel modello precedente ogni volta che attivavamo una sessione per un ruolo, guadagnavamo i permessi di quel ruolo. In questo, ogni volta che attiviamo un ruolo guadagnamo i permessi propri del ruolo, ma anche tutti quelli dei ruoli sottostanti.

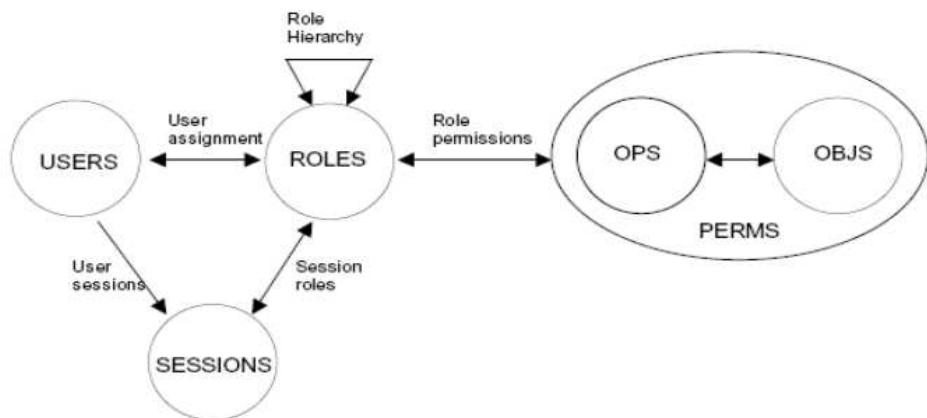


Figure 14.2: RBAC gerarchico

14.3 RBAC con vincoli SSD

SSD sta per *Static Separation of Duty*: si tratta di un modello in cui si stabiliscono a priori dei vincoli per evitare che gli utenti assumano ruoli in conflitto di interessi (ad esempio che un controllato diventi controllore).

I vincoli SSD possono essere stabiliti sia sulle relazioni **utente-ruolo** sia sulle **relazioni gerarchiche fra ruoli**.

Un vincolo SSD può essere espresso da un insieme di coppie (rs, n) , dove rs è un sottoinsieme di ruoli ed n è il massimo numero di quei ruoli che un utente può ottenere.

I vincoli tipici di questo sistema possono essere elencati come:

- **ruoli mutuamente esclusivi:** ad uno stesso utente può essere assegnato al massimo un ruolo di quelli presenti in un insieme mutuamente esclusivo. Ciò supporta il principio della separazione dei compiti;

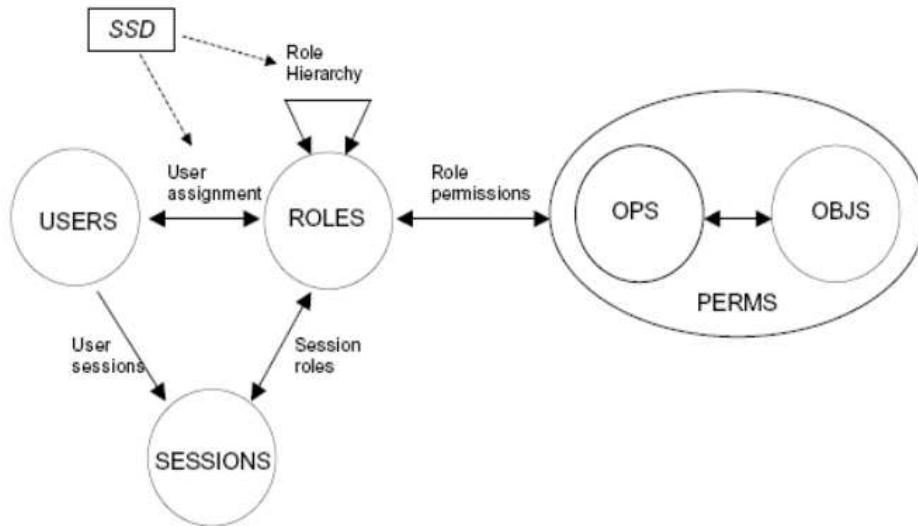


Figure 14.3: RBAC con vincoli SSD

- **permessi mutuamente esclusivi:** lo stesso permesso può essere assegnato ad al massimo un ruolo di quelli di un insieme mutuamente esclusivo (questo può essere visto come il duale del vincolo precedente e permette la distribuzione del potere);
- **cardinalità:** riguarda il massimo numero di membri appartenenti ad un ruolo (il minimo non viene generalmente utilizzato perché molto difficile da controllare);
- **ruoli prerequisiti:** ad un utente può essere assegnato il ruolo A solo se è già in possesso del ruolo B . Questo vincolo è basato sul concetto di *competenza ed appropriatezza*;
- **permessi prerequisiti:** il permesso P può essere assegnato ad un ruolo solo se è già in possesso del permesso Q . È il duale del precedente.

14.4 RBAC con vincoli DSD

In situazioni in cui i vincoli imposti staticamente risultano essere troppo restrittivi, o in cui non bastano, si ricorre a **vincoli imposti in fase di esecuzione**. È per questo che è stata introdotta una versione di RBAC con vincoli dinamici di separazione dei compiti: *DSD* o **Dynamic Separation**

of Duty. Ovviamente in questo caso è necessario il concetto di sessione che è l'unico elemento che lega l'utente al ruolo che assume.

Questa versione di RBAC supporta appieno il principio del minimo privilegio.

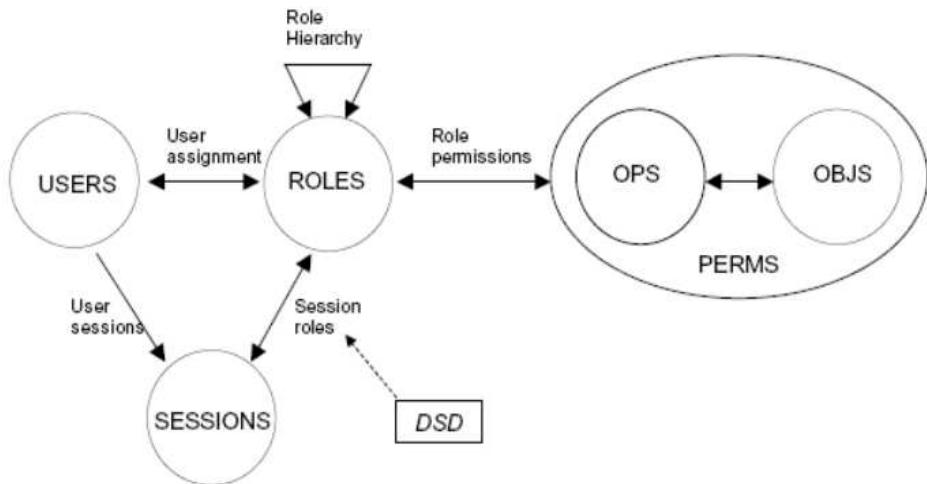


Figure 14.4: RBAC con vincoli DSD