

# Internet Security

2014-15



# **INTRODUZIONE ALLA SICUREZZA INFORMATICA**

# INTRODUZIONE ALLA SICUREZZA INFORMATICA

Gli attributi della sicurezza informatica sono:

- **Integrità**
- **Autenticazione**
- **Segretezza**

Il problema della sicurezza si espande in vari ambiti:

- **Reti:** ambito principale del corso di Internet Security.
- **Database:** Le moderne tecnologie rendono relativamente facile e poco costoso raccogliere un grande quantità di informazioni personali. Queste informazioni, nella maggior parte dei casi, vengono conservate in database. Diverse organizzazioni sono invitati, per molte ragioni, di rilasciare i dati personali che hanno acquisito. Mentre in alcuni casi è accettabile rilasciare informazioni in una forma statistica, evitando violazione della privacy, in molti altri è davvero necessario rilasciare dati specifici (chiamate anche **microdati**). Negli ultimi anni, le tecniche di anonimizzazione hanno ricevuto grande attenzione come strumento per distribuire microdati senza mettere in pericolo privacy degli utenti. In particolare, **k-anonymization** è una tecnica introdotta per proteggere l'anonymato dei dati da cosiddetti attacchi di collegamento esterni (**linking attack**).
- **Servizi innovativi:** invenzioni di nuovi servizi in rete, nuovo business, nuove startup, qualcosa che mettiamo sull'app-store. Se queste devono fare business ci saranno risorse sensibili e quindi molto probabilmente dovremo inventarci nuove misure di sicurezza per quel problema. Quindi da qui si capisce che la *sicurezza è in continua espansione*. Questo è diverso dai problemi di correttezza del software; poiché questi problemi, possiamo dire, che non cambiano: che io scriva il codice per la nuova funzionalità di facebook; che io ad esempio faccio una start up e quindi invento un nuovo servizio e lo scrivo ad esempio in PHP, probabilmente i problemi di correttezza del software non sono cambiati perché sempre codice è. Ma se con l'app che scrivo, invento un servizio innovativo, probabilmente questo aggiungerà nuovi problemi di sicurezza. Ad esempio, l'acquisto di spazio sul Cloud: c'è un account; come ci assicura che l'account abbia pagato, come ci assicura che l'account pagante abbia lo spazio e gli altri no, come si collega l'account allo spazio. Questi sono problemi caratterizzanti il proprio servizio di spazio sul Cloud.
- **Sistemi Operativi:** la gran parte delle cose che facciamo gira su un sistema operativo. Anche qui ci sono dei problemi di sicurezza molto importanti. Il problema della protezione dei dati è fondamentale. Quindi ad esempio se nel mio PC ho un sistema multiutente, devo fare in modo che un utente non legga i dati di un altro utente. Queste cose si fanno con le liste di controllo di accesso (ACL). Questo è un problema di sicurezza, ovvero il modo in cui implementarle, come do i permessi per scrivere qualcosa del genere. Si capisce subito che questo è un problema molto grande di sicurezza.

- **Programmazione:** in questo ambito si confonde il problema di correttezza e il problema di sicurezza del software. Il problema della correttezza indica l'assenza di bug; ovvero il software funziona come deve funzionare. Il problema della sicurezza è il problema che il software tratti o meno bene le risorse sensibili. Quindi se faccio un errore del tipo **null pointer dereference**, probabilmente questo è un problema di sintassi, un problema di correttezza del codice. Se faccio un errore del tipo **buffer overflow**, non sto gestendo bene la memoria. Quindi a livello di codice il problema di correttezza e sicurezza si intrecciano.

# **ESEMPI REALI E FALSI MITI**

## Esempi recenti: 1

SC Magazine > News > Poisoned YouTube ads serve Caphaw banking trojan



Danielle Walker, Reporter

[Follow @daniellewlkr](#)

February 24, 2014

## Poisoned YouTube ads serve Caphaw banking trojan

Recent YouTube visitors should be extra vigilant after ads on the website were found to be poisoned.

According to researchers at Bromium Labs, who [blogged about the threat on Friday](#), YouTube's ad network was compromised to host the Styx exploit kit.

The kit, which in recent news was pegged as [compromising online retailer Hasbro.com](#), was leveraged to spread a nasty banking trojan, called Caphaw, to users.

The Styx exploit kit spread the malware by taking advantage of a [Java vulnerability \(CVE-2013-2460\)](#), which was patched last year.



YouTube's ad network was compromised to host the Styx exploit kit, researchers found.

## Esempi recenti: 2

SC Magazine > News > Second Anonymous member sentenced for role in DDoS attack



Adam Greenberg, Reporter

[Follow @writingadam](#)

February 18, 2014

### Second Anonymous member sentenced for role in DDoS attack

The U.S. District Court, Eastern District of Wisconsin, has **sentenced** Jacob Wilkens to 24 months of probation and ordered him to pay \$110,932.71 in restitution for his role in a distributed denial-of-service (DDoS) attack against Koch Industries.

Wilkens pled guilty to intentionally causing damage to a protected computer by assisting other members of the hacktivist collective Anonymous in launching a DDoS attack on the servers of Angel Soft bathroom tissue, based in Green Bay, in February and March of 2011.

The attacks against Koch Industries were said to have lasted three days and resulted in several hundred-thousand dollars in losses.

For his role in the same attack, Christopher Sudlik was ordered earlier this month to pay the same in restitution, as well as being **sentenced** to 36 months of probation and 60 hours of community service.

## Esempi recenti: 3



Danielle Walker, Reporter

[Follow @daniellewlkr](#)

February 12, 2014

# Pre-installed security software leaves computers vulnerable to remote hijack, experts reveal

Researchers are warning that legitimate anti-theft software, impacting millions of users with the activated installation on their computers, leaves systems **vulnerable to remote hijack**.

On Wednesday, Kaspersky Lab's security team [published a report](#) on Absolute Computrace, a product developed by Austin, Texas-based Absolute Software which "allows organizations to persistently track and secure all of their endpoints within a single cloud-based console," the [product page](#) for the software says.

According to Kaspersky researchers, however, it's the fact that Absolute's tracking software is pre-installed in the firmware of laptops and desktops, and difficult to remove or disable for users, that makes its **security flaws** that much more concerning.

The report said that remote takeover of impacted systems was possible through a number of avenues.

"The protocol used by the [Computrace] Small Agent provides the basic feature of remote code execution," the report said. "The protocol doesn't use any encryption or authorization with the remote server, which creates numerous opportunities for remote attacks in a hostile network environment."

While Kaspersky hasn't seen any evidence of Computrace's weaknesses being used to carry out attacks, the researchers found that an attack on a local area network via address resolution protocol (ARP) poisoning (where a saboteur redirects all traffic from a computer running the software to their own control hub) was possible.

Another attack method could entail a domain name system (DNS) service attack "to trick the agent into connecting to a fake [command-and-control] server," the report said.

## Bit Suid

**SUID** (*Set owner User ID up on execution*) è un tipo speciale di permessi dati in un file. Normalmente in Linux / Unix, quando un programma viene eseguito, eredita le autorizzazioni di accesso dall'utente connesso. SUID è definito come dare i permessi temporanei di un utente, di eseguire un programma / file con i permessi del proprietario del file piuttosto che l'utente che lo gestisce. In parole semplici, gli utenti avranno i permessi del proprietario del file, così come proprietario UID e GID durante l'esecuzione di un file / programma / comando.

Ad esempio: consideriamo di cambiare la password di un utente (no root).

Quando cerchiamo di cambiare la nostra password useremo il comando **passwd**, che è di proprietà di root. Questo file di comando passwd cercherà di modificare alcuni file di configurazione di sistema, come `/etc/passwd`, `/etc/shadow` ecc... quando cerchiamo di cambiare la nostra password. Alcuni di questi file non possono essere aperti da tutti (tranne con apposite autorizzazioni dell'amministratore). Quindi, se rimuoviamo il BIT SUID e non diamo le autorizzazioni complete a questo file di comando passwd, non sarà possibile aprire altri file, quali file `/etc/shadow`. Senza il BIT SUID avremo qualche altro errore quando si tenta di eseguire il comando passwd. Quando invece, il comando passwd è impostato con SUID, abbiamo le autorizzazioni di root per poter eseguire questa determinata operazione.

Questa è un'operazione sensibile, perché scriviamo su uno store di sistema unico, e questa è una scelta implementativa; ma scriviamo su uno store di sistema che è sicuramente sensibile (questa non è una scelta implementativa ma un assoluto). Quindi il fatto che lo possa fare il singolo utente lo dobbiamo in qualche modo implementare. Cioè il problema non è algoritmico, ma come lo implementiamo; com'è che diamo al singolo utente la possibilità di fare questa operazione sensibile, cioè accedere a risorse condivise, a risorse di sistema. Risolviamo il tutto con il bit uid.

In conclusione; dobbiamo usare questo Bit, perché **passwd** appartiene a **root**. Se guardiamo tutta la cl, essa ha la x su g e o; quindi tutti lo possono eseguire, ma che lo possano eseguire non è sufficiente, perché lo devono eseguire da root per poter scrivere sullo store delle password; e per poter eseguire come root ci vuole il bit uid. Provando da terminale a cambiare la password di un utente semplice, appare, nei permessi che vengono specificati, una 's', che sta appunto ad indicare l'utilizzo del bit siud. Quindi in virtù della s significa che chi lancia il programma, come sancito dalla cl, lo lancerà come se fosse il proprietario del file. Uno potrebbe dire ma la 'x' c'è e quindi gli altri la possono eseguire, ma il proprietario è root; quindi il completamento dell'operazione con la scrittura della nuova password lo deve fare root, e affinché questo lo possa fare qualsiasi altro utente serve proprio il bit Suid.

## Esempio di Trojan Horse

- Ogni file Unix ha un proprietario e un gruppo
- Normalmente un programma riceve i permessi dell'utente (provare whoami) che lo lancia
- Si veda a chi appartiene il programma passwd
- Come potrebbe un qualunque utente cambiarsi la password?
- Si utilizza il bit SUID (Set owner User ID up on execution): settato, programma lanciato con permessi del proprietario
- Ipotesi: la vittima riceva il seguente Trojan sotto forma di file eseguibile chiamato ls.
- Danno riscontrabile dalla vittima?

### Esempi

```
cp /bin/sh /tmp/.xxsh  
chmod u+s,o+x /tmp/.xxsh  
rm ./ls  
ls $*
```

- Aggiungere utente victim
- Utente giamp può fare touch prova.txt nella home di victim?
- Costruire detto ls nella home di victim
- Renderlo eseguibile
- Eseguirlo
- Utente giamp può lanciare /tmp/.xxsh coi permessi di victim!

### Soluzione

Directory “.” eliminata dal path di default, mentre in passato era fondamentale!

# Strumenti generali della sicurezza

Gli strumenti generali per la sicurezza sono i seguenti:

- **Crittografia:** è uno degli strumenti più importanti.
- **Politiche:**
  - **Ex1.** La politica di sicurezza che sta alla base dello sviluppo di un sistema operativo. Se consideriamo le ACL, esse con quale criterio sono state realizzate. Il criterio che ci sta dietro è una politica di sicurezza.
  - **Ex2.** Firewall. Il criterio che ci sta dietro un firewall è un criterio di sicurezza
- **Conoscenza, possesso e biometria:**
  - **Conoscenza:** L'autenticazione è uno dei problemi principali della sicurezza. Il tentativo di autenticarci è il tentativo di discriminare una persona rispetto un'altra. Il punto è che questa discriminazione non esiste in natura. Diciamo che è un qualcosa inventata da noi, nel senso che chi sa una cosa lo possiamo discriminare da chi non la sa. Questo è il concetto di autenticazione.
  - **Possesso:** ancora più classista; ad esempio chi esibisce un distintivo lo faccio entrare e chi non la esibisce no. Ma l'autenticazione, rientra nella categoria di autenticazione basata sul possesso. Ad esempio il tesserino mensa. Quindi il possesso viene utilizzato come artifizio per discriminare fra gli uomini.
  - **Biometria:** anche questo potrebbe essere utilizzata per discriminare. Le impronte digitali si dicono che siano diverse tra una persona all'altra, e quindi sembra che sia un discriminante tra gli uomini connaturale. Quindi sembra che abbiamo trovato un modo di autenticare un individuo (non utilizzando il "possesso"). Ma questa, tende ad essere un'arma poco debole; perché se l'individuo non vuole condividere le sue credenziali, la password non la dice; ma se consideriamo le impronte digitali queste potrebbero essere lasciate da qualsiasi parte in modo del tutto involontario dall'individuo.
- **Programmi di protezione;** quali firewall (rilevamento delle intrusioni), protocolli di sicurezza (SSL è il protocollo più diffuso in rete)
- **Sensibilizzazione dell'utente;** social engineering o social tecnologici della sicurezza. Riguarda il rapporto tra il computer e l'umano.

## Limiti della crittografia

La crittografia è un ramo della matematica. Quindi essa è molto basata su teoremi; bisogna stabilire determinare ipotesi e giungere a conclusioni. In pratica dovremmo applicare queste soluzioni, ma non è sempre immediato, per vari motivi, per esempio vogliamo fare un inscription di una smart card. Si può fare, però la smart card ha dei limiti computazionali; quindi ad esempio dobbiamo diminuire la lunghezza della chiave per fare un inscription. Da qui si capisce che garanzie formali su quel certo ecosistema magari dovevano essere basate su chiavi di lunghezza superiore a quelle che invece possiamo gestire nella pratica. Un altro esempio dei limiti della crittografia potrebbe essere quelle riguardanti le leggi nazionali; che spesso limitano l'import e l'export di crittosistemi.

## Limiti dell'uso di password

I problemi principali delle password sono le seguenti:

1. **Scelta di una buona password**: una buona password è una password difficile da indovinare da parte di un attaccante.
2. **Mnemonicità di una buona password**: la password deve essere mnemonica.
3. **Attacchi dizionario**: se la password è talmente banale di essere una parola di un dizionario; qualunque sistema per violare una password, per prima cosa, si passa in rassegna un dizionario.

Una discreta difesa contro il “*dictionary attack*” potrebbe essere quella di non usare parole contenute nel dizionario, per esempio usare parole appartenenti a qualche dialetto italiano, oppure trovare un metodo mnemonico per ricordare la password scelta, ad esempio le iniziali di una frase che è facile ricordare:

*Io Sono un Tifoso della Squadra A – potrebbe risolversi nella password: ISuTdSA,*

come si evince dall'esempio l'uso delle lettere minuscole e maiuscole complica la scoperta. L'abbinato uso di numeri e punteggiatura, inoltre, complicherebbe ulteriormente la scoperta per qualsiasi software.

Questo non significa, che la nostra password possa considerarsi una: “strong password”, specie se la password scelta viene usata per più sistemi. Per questo si suggerisce di usare la stessa tecnica per formare password differenti per ottenere l'accesso ad ogni singolo sistema.

4. **Attacchi statistici**
5. **Periodicità di una password**: una password periodica è una password che deve cambiare periodicamente. Perché se una password non viene cambiata periodicamente non sarebbe una buona cosa?

- No. Perché c'è più tempo per attaccare la password

**6. Riutilizzo o utilizzo multiplo di una password:** molti utenti usano la stessa password per servizi diversi. Cioè lo stesso utente può usare la stessa password per due servizi differenti, dove uno è più importante e l'altro meno. Un servizio meno importante, ad esempio un portale di notizie ecc.., non mi garantisce che la mia password sia trattata come si deve, che attraversi la rete in maniera sicura, che sia inserita in un database in maniera corretta. Un ricercatore, studiò quante di queste situazioni sono presenti. Notò, appunto, che la gente mette una password su un servizio non molto importante e spesso questa password viene persa. Solo che queste persone sono così tonte che hanno utilizzato la stessa password su un servizio sensibile. Questo ricercatore ha trovato tanti riscontri di questo genere.

C'è un molteplice utilizzo della stessa password. Il motivo è descritto nel punto successivo.

**7. Numerosità delle password per utente:** se abbiamo tante password, dobbiamo trovare un modo per come tenerle. Più password abbiamo e più è difficile ricordarle. Quindi uno dei problemi è: come riusciamo a tenere in modo sicuro tutte queste password?

Ricordare password difficili (definite nel punto 2) è complicato.

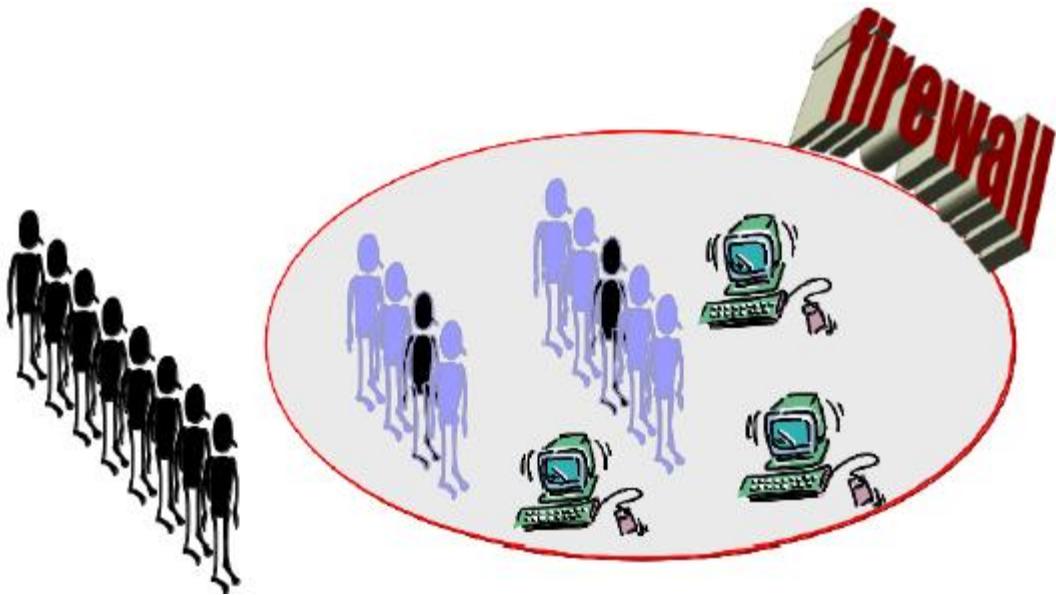
## Firewall

Un firewall è un sistema hardware-software dedicato alla difesa perimetrale di una rete che agisce filtrando il traffico di pacchetti entranti e/o uscenti.

Quindi esso rafforza la sicurezza della rete interna preservandola dai pericoli della rete esterna. Nella maggior parte dei casi ciò significa proteggere la propria LAN domestica o aziendale da attacchi provenienti da Internet, ma l'uso di un firewall si può estendere anche ad altri scenari. Non sempre infatti la rete esterna dalla quale guardarsi è il world wide web, ma i pericoli potrebbero provenire da uno dei computer presenti nella stessa sottorete. Ad esempio un utente esperto potrebbe decidere di eseguire attacchi verso il server in modo da poter controllare l'intera rete; oppure, più semplicemente, un virus eseguito su uno dei client (magari introdotto a causa di un floppy disk infetto) potrebbe tentare di diffondersi nell'intera rete. In questi casi un firewall posto tra il server e i client renderebbe più tranquilla la vita dell'amministratore.

Il concetto di firewall si affermò con gli inizi degli anni '90, quando la diffusione di Internet aumentò notevolmente il problema della sicurezza delle reti. Inizialmente i firewall non erano nient'altro che programmi implementati e compilati dagli stessi amministratori con lo scopo di controllare la provenienza dei pacchetti. Adesso invece un buon firewall può svolgere diversi compiti.

Un firewall lo possiamo considerare così:



Sebbene un firewall sia un componente fondamentale per la sicurezza di una rete, **questo non vuol dire che la protezione dell'intera rete possa essere delegata unicamente al firewall**. Ci sono delle problematiche che dovranno essere assegnate ad altri software, perché non di competenza del firewall. Eccone alcune.

- **Protezione da attacchi interni.** Una volta che il “muro” del firewall è stato oltrepassato, quest’ultimo non è più di alcuna utilità. Idem se l’attacco nasce all’interno della rete protetta dal firewall, dobbiamo tenere a mente che il tipo di protezione che esso ci fornisce è perimetrale, quindi tutto ciò che è interno all’area è escluso dal filtraggio. In questi casi per cautelarsi sarebbe opportuno stabilire in ogni computer della rete interna dei permessi restrittivi da associare ai vari utenti, oppure inserire un ulteriore firewall a protezione del server.
- **Social engineering.** Usato per indicare coloro che telefonano agli impiegati spacciandosi per membri della sicurezza, così da estorcere preziose informazioni quali indirizzi ip, password, ecc. In questo caso la soluzione è semplicemente quella di istruire gli impiegati a non rispondere a tali domande se non in casi eccezionali.
- **Integrità dei dati.** Nonostante i moderni firewall controllino costantemente la presenza di virus nei pacchetti, è impensabile su reti di grandi dimensioni chiedere al firewall di controllare tutti i pacchetti nella ricerca di virus. Ancora più difficilmente possono rilevare i cavalli di Troia, che non cercano di diffondersi verso altri file o computer, ma che si limitano ad aprire una back door nello stesso computer. Per venire a capo di queste situazioni è necessaria la presenza di un buon antivirus installato in tutte le macchine.
- **Cattiva configurazione.** Un firewall non è capace di distinguere da solo ciò che va bloccato e ciò che invece va accettato. La qualità del firewall deve essere supportata dalla qualità della sua configurazione da parte dell’amministratore. La maggior parte dei firewall, ad esempio, non sono capaci di gestire situazioni in cui i pacchetti sono molto frammentati. In questo caso è necessario aver precedentemente esplicitato una regola per tale situazione.

## Definizione di sicurezza per punti:

1. Non un prodotto ma un processo
2. **Anello più debole di una catena:** il nostro sistema sarà sicuro tanto quanto sarà sicuro l'anello più debole della catena. Insomma se la catena ha un solo anello debole non serve a niente. Quindi dobbiamo mettere in sicurezza tutto. Solo il firewall non basta, quindi ad esempio dovremmo aggiungere anche un sistema di autenticazione, un sistema di controllo di accesso, ecc..
3. **Espressa da che cosa:** ogni volta che diciamo che quel determinato sistema è sicuro, dobbiamo specificare da che cosa è sicuro. Questo è fondamentale perché una volta realizzato un sistema, un protocollo, o altro, e diremo che è sicuro da determinati attacchi; il sistema sarà sicuro solo da questo aspetto e meno sicuro da altri. Quindi **i nostri sistemi di sicurezza sono limitati ai tipi di minacce che consideriamo.**
4. **Sempre soggetta all'analisi costi/benefici dell'attaccante:** ciò che regolarizza il punto 3) è un'analisi dei benefici dell'attaccante.
5. **Si realizza in pratica mediante livelli di sicurezza:** cioè quel determinato sistema ha un buon livello di sicurezza, ma non ne ha uno migliore; per cui un attaccante "studierà" questi aspetti.

## Rischi base per la sicurezza

- **Complessità del singolo sistema da mettere in sicurezza (browser, comunicazioni):** il nostro computer è un sistema complesso, perché non solo abbiamo il problema delle porte, ma abbiamo anche il problema di un software che può arrivarci come trojan perché l'utente si scarica l'allegato; abbiamo anche il problema del browser. Esso è uno degli applicativi più complessi, perché gestisce tantissime cose. Lavora sulle porte importanti, e uno dei problemi storici era quello di capire che permessi dare a questo processo (browser). Se ad esempio uno col browser voleva cambiare la password, si dovevano dare i permessi di root. Originariamente era così. Questo non va bene.
- **Combinazione di sistemi (sicurezza punto-punto come sicurezza di molteplici sistemi):** una BTN è un esempio di sicurezza punto-punto, vuol dire che tutto quello che ci sta di mezzo non ci può attaccare.
- **Predisposizione ai bug (circa uno ogni mille linee di codice)**
- **Proprietà emergenti (data austerity, prevenire il double-spending)**
- **Interazione con l'uomo (human as the weakest link in the chain)**

# Rischi digitali per la sicurezza

## 1. Automazione offensiva

- 1 Microfurti (*limare 1 centesimo da ogni transazione VISA*)
- 2 Violazioni (quasi) intracciabili (*app crash, reboot*)
- 3 Privacy a rischio (*i nostri dati su numerosi database*)

## 2. Assenza distanza

- 1 Grazie alle reti (*Internet non ha confini*)
- 2 Tutti i criminali contro il nostro sistema (*adolescente che scarica exploit*)
- 3 Non bastano le leggi nazionali a proteggerci (*attacco da paese x senza accordi di estradizione*)

- **Automazione dell'offensiva:** se scriviamo un programma malevolo, lo possiamo atomizzare in maniera semplice. Tempo fa una cosa del genere si chiamava **microfurti**. Immaginiamo di limare un centesimo da una transazione; con semplicissimi programmi si poteva automatizzare l'appropriazione di un centesimo (se facevamo tante appropriazioni avevamo tanti di questi centesimi). Le violazioni sono quasi intracciabili. Quando si parlerà di **intrusion detection**, tracciare l'intruso non è facile. Gli strumenti che abbiamo a disposizione sono pochi. Quindi ad esempio, possiamo guardare i processi attivi; killiamo (uccidiamo), quel processo che per noi è sospetto ovvero è un possibile intruso. Ma poiché non abbiamo capito da dove è entrato questo intruso, dopo qualche secondo lo possiamo ritrovare con noi.
- **Assenza di distanza:** per mettere in sicurezza quella determinata porta, lavoriamo su di essa sapendo che l'attaccante è là dietro e non ci può attaccare. Col mondo digitale tutto ciò non è più vero. Gli attaccanti stanno dappertutto.

Approfondire con il libro: bruce schneier sicurezza digitale

### 3. Propagazione tecniche

- 1 Rapidità tecniche (*hacker pubblica attacco, crack turco per editor*)
- 2 Diventare offensivi non implica abilità (*scaricato script per DoS, trovato codice rubato*)

### 4. Difficoltà reazione

- 1 Semplici stranezze (*certo che ti ho mandato l'email, giuro!*)
- 2 Reali furti (*transazione di €100 sul mio conto ma... non è mia!*)

- **Propagazione delle tecniche:** alcuni mettono gli exploit sulla propria pagina, fin quando non li beccano. Un exploit è un termine usato in informatica per identificare un codice che, sfruttando un bug o una vulnerabilità, porta all'esecuzione di codice non previsto, il cui risultato può portare all'acquisizione di privilegi maggiormente elevati di una macchina informatica.

#### Metodologia d'attacco

- 1 Studiare il sistema target (*port scanning*)
- 2 Cercarne (in rete?) potenziali punti deboli (*deamon flaw*)
- 3 Disegnare (o scaricare!) eseguibili per verificare i punti deboli (*exploits*)
- 4 Goto 1

#### Metodologia di difesa

- 1 Installare strumenti di difesa (*firewall, IDS*)
- 2 Aggiornare il sistema (*updates, security patch*)
- 3 Monitorare il sistema
- 4 Goto 1

- **Metodologie di attacco:** l'attaccante ci scansiona continuamente. Studia il sistema target. L'attaccante studia da chi è mantenuto quel sistema; se quest'ultimo se ne intende di queste cose oppure no.  
*Non ci occuperemo della scrittura di exploits.*
- **Metodologia di difesa:** gli strumenti che utilizziamo per difenderci da attacchi esterni. Esse riguarderanno la scrittura di regole furbe per un firewall; la configurazione di un sistema di intrusione.  
Altri strumenti: IDS, firewall, antivirus, le politiche.

# **PROPRIETA', ATTACCHI E ATTACCANTI**

## Attacchi

Il parallelo col mondo reale è semplice, perché i problemi di sicurezza ci sono anche nel mondo reale. Pensiamo alla sicurezza delle banche.

Gli attacchi li potremo classificare in base *all'obiettivo*.

Gli attacchi sono di natura svariata (tante cose noi potremmo chiamare attacco).

Per esempio:

- **Guadagno di privilegi superiori:** se un utente limitato riesce a fare delle operazioni con i privilegi di root, è un attacco.
- **Attacchi di segretezza**
- **Mancato recapito:** pensiamo alla posta elettronica. Ecco perché esiste la posta elettronica certificata.
- ...

Una definizione che possiamo dare alla sicurezza è la seguente:

**sicurezza = prevenzione attacchi + corretto funzionamento del sistema**

Si potrebbe pensare che il computer più sicuro è quello che non è collegato alla rete. Quindi un computer sicuro è un computer spento. Questo riguarda la prima parte della definizione data pocanzi di sicurezza (prevenzione attacchi). Ma ciò è una estremizzazione. Cioè ci vuole anche la seconda parte della definizione; ovvero il sistema deve funzionare, perché il computer spento sarà inviolabile ma non funziona (corretto funzionamento del sistema).

**Definizione: Sicurezza vuole dire protezione da attacchi pur mantenendo il funzionamento (l'usabilità) del sistema.**

## Attacco reale: furto (perdita) di dispositivi

### Teoria: contromisure

- ① **Attacco fondamentale:** accesso al sistema
  - Contromisura: autenticazione al sistema
    - Laptop: password, impronta
    - Smartphone: pin, viso, pattern, impronta, anche combinate con cancellazione memoria
- ② **Attacco successivo 1:** uso funzionalità di sistema
  - Contromisura: autenticazione alla funzionalità
    - Laptop: browser richiede autenticazione al server
    - Smartphone: app richiede autenticazione al servizio
- ③ **Attacco successivo 2:** acquisizione di dati sensibili
  - Contromisura: crittografia
    - Laptop: codifica delle password o dell'intero file system
    - Smartphone: idem (?)

Se perdiamo un computer, un telefonino o altro, potremmo subire un attacco. Bisogna pensare a delle contromisure per questo problema:

#### 0. Il problema fondamentale è l'accesso al sistema

- **Contromisura: autenticarsi al sistema.** Poiché ci potrebbe essere il pericolo che mi rubino il dispositivo allora richiedo autenticazione
  - Sui laptop possiamo inserire password oppure verificare tramite impronte
  - Sui smartphone abbiamo altri metodi: pin, impronta. Sui sistemi Android c'è pure il riconoscimento del viso.

*Vale la regola, che più è sensibile l'obiettivo e più è probabile che il dispositivo viene violato.*

Sicuramente tutte le soluzioni che esistono di sicurezza non ci garantiscono che il sistema sia *assolutamente* sicuro.

1. Una volta che abbiamo accesso al sistema, possiamo usare tutte le funzionalità. Quindi supponendo che abbiamo perso un nostro dispositivo, e che riescono a scoprire la password, vogliamo vedere se l'attaccante riesce ad usare le funzionalità. L'utente potrebbe risolvere questo problema mettendo un **autenticazione alle funzionalità**, cioè ogni qualvolta che l'attaccante voglia usare una funzionalità della macchina, bisogna mettere un'altra password. Per esempio supponiamo di decodificare la home. Per decodificarla bisogna mettere un'altra password. Quindi dobbiamo autenticarci per l'ennesima volta. Questa capita ad esempio in alcuni servizi di rete; capita di autenticarci, e non basta essere già autenticati con l'attuale utente. Quindi stiamo definendo sia **l'autenticazione d'accesso** che **l'autenticazione alle "funzionalità"**.
2. Ulteriore attacco: supponiamo che siamo riusciti sia ad accedere al sistema e sia ad accedere alle funzionalità di sistema. Se questo si verifica possiamo **acquisire dati sensibili** (già riuscire a scoprire la password è un dato sensibile). L'utente, per risolvere questo problema, potrebbe codificare tutto. Ad esempio sui laptop potrebbe codificare l'intero file system. Quindi se un utente non ha codificato quest'ultima parte, allora potrebbe subire gli attacchi descritti nei punti 0-1-2.

Quindi un attaccante potrebbe esser riuscito ad entrare nel sistema, trovare la password per accedere alla posta (funzionalità del sistema). Le password sono tutte conservate. Le password utente sono scritte con un certo livello di protezione, ovvero sono in versione Hash. Anche questo sistema di protezione è violabile. **L'attacco 1 e 2 non sono la stessa cosa.** L'attacco 1 è: io attaccante voglio usare le funzionalità di sistema; se trovo la password attacco. L'attacco 2 indica che io voglio scoprire la password che ha usato la vittima. Questo potrebbe servire all'attaccante per fare **social engineering\***. Quindi in questo caso, se viene scoperta la password della posta, probabilmente la vittima l'avrà usata anche in altri servizi (la situazione diventa critica).

\***Social engineering:** Nel campo della sicurezza informatica, l'**ingegneria sociale** (dall'inglese *social engineering*) è lo studio del comportamento individuale di una persona al fine di carpire informazioni utili.

Poiché l'utente è attento a tutto questo, decide di codificare tutto. Codifica l'unità, codifica le password per l'accesso al sistema, codifica le password di accesso ai servizi via browser. In questo modo ho risolto tutti i problemi descritti prima? Nella pagina precedente sono state descritte alcune contromisure. Ma è "teoria". In pratica queste contromisure potrebbero fallire.

### Pratica: contromisure fallite!

#### 0 Attacco fondamentale: accesso al sistema

- Contromisura: autenticazione al sistema
  - Laptop: riambientazione memoria di massa, boot da dispositivo esterno
  - Smartphone: spesso disabilitata per ragioni di usabilità

#### 1 Attacco successivo 1: uso funzionalità di sistema

- Contromisura: autenticazione alla funzionalità
  - Laptop: browser registra password
  - Smartphone: app registra password

#### 2 Attacco successivo 2: acquisizione di dati sensibili

- Contromisura: crittografia
  - Laptop: riluttanza verso la codifica, quindi accesso a password in chiaro, copia di documento, di dati bancari, etc.
  - Smartphone: idem (?)

0. Attacco fondamentale: accesso al sistema; esistono modi per accedere al sistema senza conoscere la password utente, ad esempio usando boot da dispositivi esterni, ecc...

1. Uso di funzionalità di sistema. I browser si memorizzano le password. Se consideriamo gli smartphone, nei playstore quando acquistiamo qualcosa viene registrata la carta di credito. Bisogna stare molto attento nell'uso di strumenti di questo tipo. In queste situazioni l'attaccante, ha tutti gli strumenti belli pronti per poter attaccare.

Spesso il mercato favorisce l'usabilità a discapito della sicurezza.

2. Uso di crittografia. Consideriamo una macchina non tanto nuova, di qualche anno fa. Messa alla prova, la macchina rallenta. Quindi le prestazioni diventano un'accezione dell'usabilità; poiché rallenta non usiamo gli strumenti crittografici.

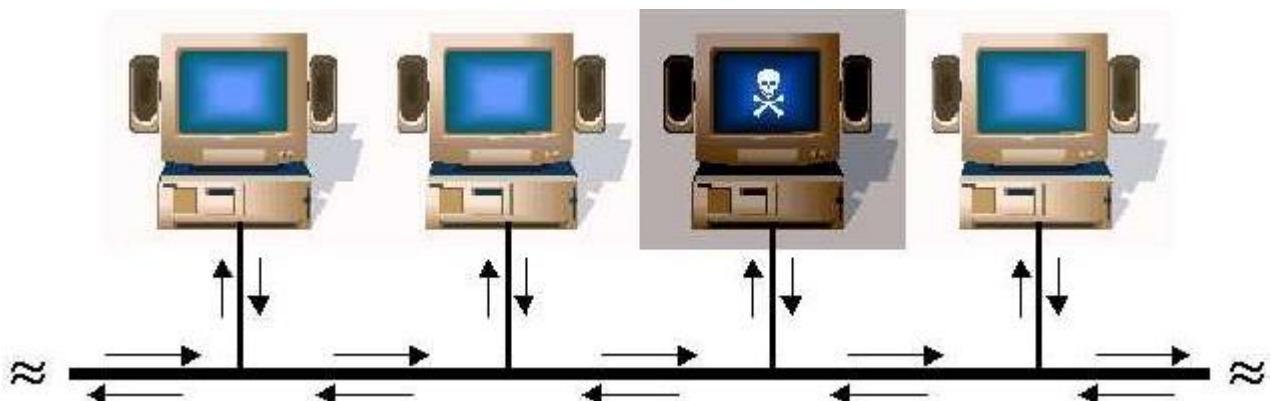
**Concludendo:** riluttanza verso la codifica e quindi il famoso tizio che ha perso il dispositivo si è beccato gli attacchi 0, 1 e 2. **PESANTE!!!!**

## Attacco reale: Pirateria digitale

La pirateria è un attacco.

- Il furto di proprietà intellettuale è ancora un problema aperto. A metà degli anni '90 c'è chi faceva ricerca sulla steganografia. L'idea è quella di incorporare un file con altre informazioni.
- Furto di identità: spacciarsi per qualcun altro
- Furto di marchio: più che dal punto di vista tecnologico, sono protette dalla giurisprudenza.

## Attacco reale: sniffing



Esistono diversi tipi di minacce e attacchi a cui una rete può essere soggetta. Tuttavia gli obiettivi generali sono gli stessi: essi mirano infatti a compromettere la riservatezza, l'integrità o la disponibilità dei dati, del software e dell'hardware.

Principali metodi di attacco:

- **Spoofing**
- **Sniffing**
- **Phishing**
- **Keylogger**

Sniffing: Attività di intercettazione passiva dei dati che transitano in una rete.

Può essere applicato sia per scopi legittimi che illeciti:

- individuazione di problemi di comunicazione
- individuazione di tentativi di intrusione
- intercettazione fraudolenta di password o altri dati sensibili

Gli *sniffer* offrono strumenti di analisi che analizzano tutti i pacchetti di una connessione TCP per valutare il comportamento del protocollo o per ricostruire lo scambio di dati tra le applicazioni.

# Tassonomia di attacchi



Abbiamo vari tipi di attacchi:

- **Attacchi criminali**
- **Attacchi di privatezza**
- **Attacchi a scopo pubblicitario**
- **Attacchi basati sui sistemi legali.**

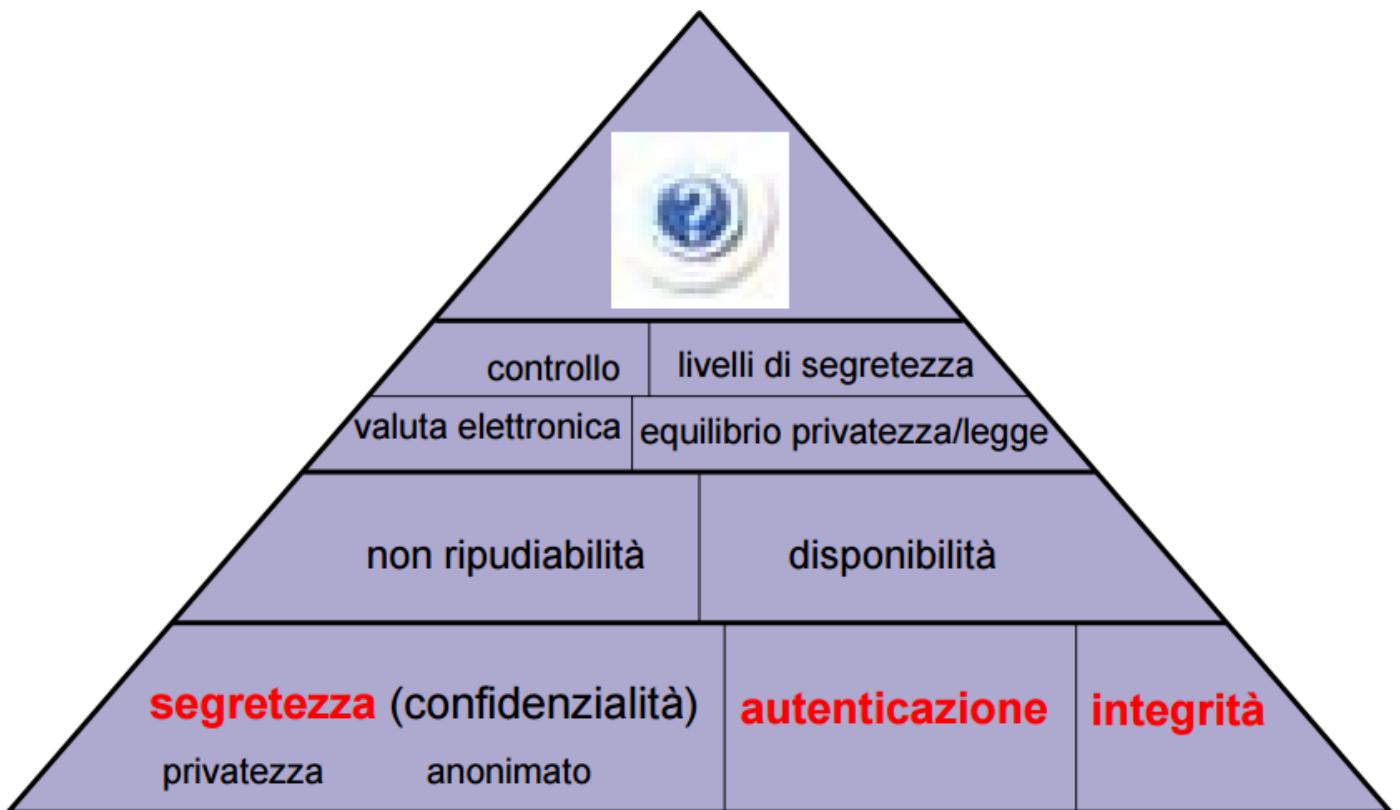
Gli ultimi due hanno degli effetti. Per gli attacchi a scopi pubblicitari, come esempio, possiamo considerare la negazione del servizio. Per gli attacchi basati sui sistemi legali, come esempio possiamo considerare i dati registrati negli istituti bancari.

Fra gli attacchi criminali abbiamo, frodi, furti di marchi registrati, furto d'identità, furto di proprietà intellettuale, attacchi distruttivi. Il furto monetario sulle transazioni bancarie è in generale gli attacchi distruttivi che tendono ad inchiodare il sistema.

Le violazioni di privatezza passano dalle violazioni sui database all'analisi del traffico. L'analisi del traffico è l'analisi di informazioni del traffico possibilmente incomprensibili. Ex. Faccio sniffing sulla LAN e scopro che 3 giorni prima dell'esame c'è un forte traffico di email tra un docente e uno studente.

**APPROFONDIRE BENE DAL LIBRO.**

# Proprietà di Sicurezza



Alla base troviamo: **segretezza** (confidenzialità), **autenticazione** e **integrità**.

Il **non-riudio** indica l'impossibilità di dire falsità, di negare la propria partecipazione in una sessione di comunicazione; qualcosa del tipo: posta elettronica certificata.

La **disponibilità** ha a che fare con il fatto che se abbiamo un conto bancario online e lo troviamo down, non piace la cosa. Non è chiaro che nesso ci sia tra la disponibilità e la sicurezza in generale. Non è perché il sito è down è sicuro. Magari è down per manutenzione. Però la disponibilità del servizio è considerata, per ragioni *socio-tecniche*, un fattore di affidabilità.

Salendo sopra di livello, troviamo proprietà più specifiche, ovvero ciò che ha a che fare con la valuta elettronica (pensiamo a **bitcoin**).

**Controllo:** si riferisce al controllo di accesso.

**Livelli di segretezza:** per i sistemi mandatori, sistemi militari.

**Equilibrio privatezza/legge:** privatezza dell'individuo verso ciò che la legge vuole fare.

## Segretezza (confidenzialità)

### Definizione

L'informazione non sia rilasciata ad entità non autorizzate a conoscerla

Quando si parla di segretezza c'è un problema di condivisione. Invento un segreto e per poter essere sfruttato deve essere **condiviso**. Questo è quello che la definizione dice: "è segreto se lo conoscono solo quelli giusti".

**Ex1.** Parola d'ordine: "tanti saluti". Se la parola d'ordine non è condivisa tra il soggetto e il ricevente non serve a niente. **Quindi segreto nel senso scolastico del termine non esiste.**

Ogni qualvolta che si parla di segretezza stiamo **implicitamente** individuando l'insieme degli eventi per un segreto.

Il problema principale sta nel come recapitare questo segreto agli eletti che devono conoscerlo.

Per risolvere questo problema possiamo usare alcuni strumenti:

- **Crittografia:** protocollo crittografico (**SSL, IPSec**)
- **Steganografia:** protocollo steganografico.

**Nota:** La steganografia realizza un tipo di sicurezza che è la segretezza, perché ciò che codifichiamo nei bit meno significativi (ad esempio supponendo di lavorare su immagini), non è semplice da individuare e di conseguenza un utente comune non se ne accorge di tutto ciò. Quindi con questa tecnica si nascondono informazioni e di conseguenza si ottiene segretezza.

# Autenticazione

## Definizione

Le entità siano esattamente chi dichiarano di essere

Per autenticazione intendiamo essere autenticato, essere riconosciuto. Quando un utente è autenticato gli si apre tutto un mondo. Quindi essere autenticato non è fine a se stesso, vuol dire che ci sono una serie di funzionalità alle quali nell'essere riconosciuto dà la possibilità di usarle.

Si hanno vantaggi combinando autenticazione e segretezza: dopo che un utente si autentica, ha accesso a qualcosa che gli altri non possono avere. Quindi, in questo modo, quell'utente diventa “meglio” di tutti gli altri utenti che non conoscono quel segreto. Autenticazione e segretezza fatti a cascata realizzano questo schema sicuro.

**Osservazione:** In generale, uno che dichiara di essere, per esempio, Obama, non è detto che poi possa dimostrare questa affermazione. Cioè se non c'è uno strumento per autenticare (per dire chi sono) non c'è autenticazione.

L'autenticazione si può fare con:

- **Conoscenza:** solo chi conosce quel segreto può fare determinate cose
- **Possesso:** solo possiede strumenti particolari (smart card, ecc...) potrà fare determinate cose
- **Biometria:** solo chi ha inherentemente, biologicamente, quell'insieme di caratteristiche biometriche (in questo caso) potrà fare determinate cose.

# Integrità

## Definizione

L'informazione non sia modificata da entità non autorizzate

Integrità delle informazioni: il sistema deve impedire la alterazione diretta o indiretta delle informazioni, sia da parte di utenti e processi non autorizzati, che a seguito di eventi accidentali. Anche la perdita di dati (per esempio a seguito di cancellazione o danneggiamento), viene considerata come alterazione.

Un soggetto non autorizzato entra in possesso di una componente del sistema, la modifica e la introduce di nuovo nel sistema. Questo è un **attacco all'integrità**.

# Privatezza

## Definizione

Diritto di un'entità di rilasciare o meno i propri dati personali ad altre entità

La privatezza sembra simile alla segretezza, ma a differenza di quest'ultima, la privatezza parla di diritto. Questa definizione è basata sulle leggi Europee (in America questo diritto è molto discusso). **Quindi la privatezza è un diritto di segretezza dei propri dati personali.** La privatezza, a differenza della segretezza, tratta un ben preciso tipo di dati, i dati personali, dette **informazioni personali statiche**. Esistono anche le informazioni personali dinamiche, ad esempio i messaggi su WhatsApp, le ricerche su Google che facciamo, ecc... .

## Osservazioni:

- **Privatezza come diritto di segretezza**
  - La password dei nostri laboratori è segreta, non privata!
    - C'è una policy che accettiamo quando ci viene data la password. La policy dice cosa possiamo fare con la password, ovvero che non la possiamo dare a nessuno. La password, in questo caso, è segreta perché non abbiamo il diritto di rilasciarla, quindi non abbiamo un diritto sulla segretezza della password. Se diamo la password a qualcuno stiamo violando quanto scritto nella policy.
    - Consideriamo la data di nascita. Essa è privata. Se la divulgiamo stiamo esercitando il nostro diritto di segretezza di rilasciarla.
- **Privatezza a protezione dell'individuo**
  - Bombardamento pubblicitario, importanza delle informazioni mediche
- **Elemento fondamentale della democrazia:**
  - Privatezza del voto, dei contenuti della propria abitazione, delle conversazioni telefoniche
- **Diritto in UE, non in USA**
  - Dati della registrazione su un sito Italiano o su Facebook, dati statici vs dinamici
- **Temporalità**
  - Database clienti non va protetto per sempre
    - Se ci registriamo su un servizio, c'è un limite nel mantenere i nostri dati privati (16 anni); un tempo relativamente breve. Quindi i nostri dati potrebbero essere, allo scadere di questo tempo, venduti.

## Alcune misure

- Isolamento sociale (!)
- Consenso a policy

## Anonimato

### Definizione

Diritto dell'iniziatore di una transazione di rilasciare o meno la propria identità ad altre entità

L'anonimato è un particolare tipo di privacy che riguarda la propria identità.

Tutti browser danno la possibilità di poter effettuare navigazione anonima (nascosta, incognita). Ma non significa che nessuno sa quello che stiamo visualizzando. Da questi strumenti ci aspettiamo di non lasciare tracce di quello che facciamo. Ma non funziona così; anzi tutto l'opposto. Questo tipo di anonimato non lascia tracce sulla nostra macchina, ma al provider viene detto tutto, come se fosse una navigazione normale (IP, pagine che visualizziamo). Lo possiamo considerare come un animato al contrario (locale).

Per **pseudo-anonimato** si intende un **anonimato “computazionale”**, cioè con un calcolo posso risolvere l'associazione nickname - utente privato (quindi risolvere in qualche modo l'anonimato).

**Ex: TOR:** Tor è una rete di tunnel virtuali che permette a persone e gruppi di aumentare la privacy e la sicurezza su Internet. Consente inoltre agli sviluppatori di software di creare nuovi strumenti di comunicazione con caratteristiche privacy incorporate. Tor fornisce le basi per una gamma di applicazioni che consentono alle organizzazioni e agli individui di condividere informazioni sulla rete pubblica senza compromettere la loro privacy. La soluzione proposta da Tor è:

- Creare una rete di nodi parallela a Internet per l'instradamento dei pacchetti
- La rete di tor funziona come una scatola nera (black box): i pacchetti che vi entrano scompaiono e appaiono “auto magicamente” all'uscita, dopo aver percorso un viaggio all'interno della rete parallela
- L'idea è quella di raggiungere la destinazione cancellando le tracce che ci lasciamo dietro, in modo da rendere impossibile l'analisi del traffico

Tor sta per “**router a cipolla**”. Si chiama così perché ogni nodo conosce solo che un pacchetto gli arriva dal nodo a monte e devo consegnarlo al nodo a valle. I nodi intermedi non possono leggere il contenuto del payload di partenza. In questo modo riusciamo a fuggire dalle tecniche di analisi del traffico in quanto non è possibile risalire agli attori del dialogo senza riuscire a leggere TUTTO il traffico che viaggia all'interno della rete di tor e, anche in questo malaugurato caso, non si avrebbe la certezza matematica dell'individuazione dei partecipanti ma solo una approssimazione.

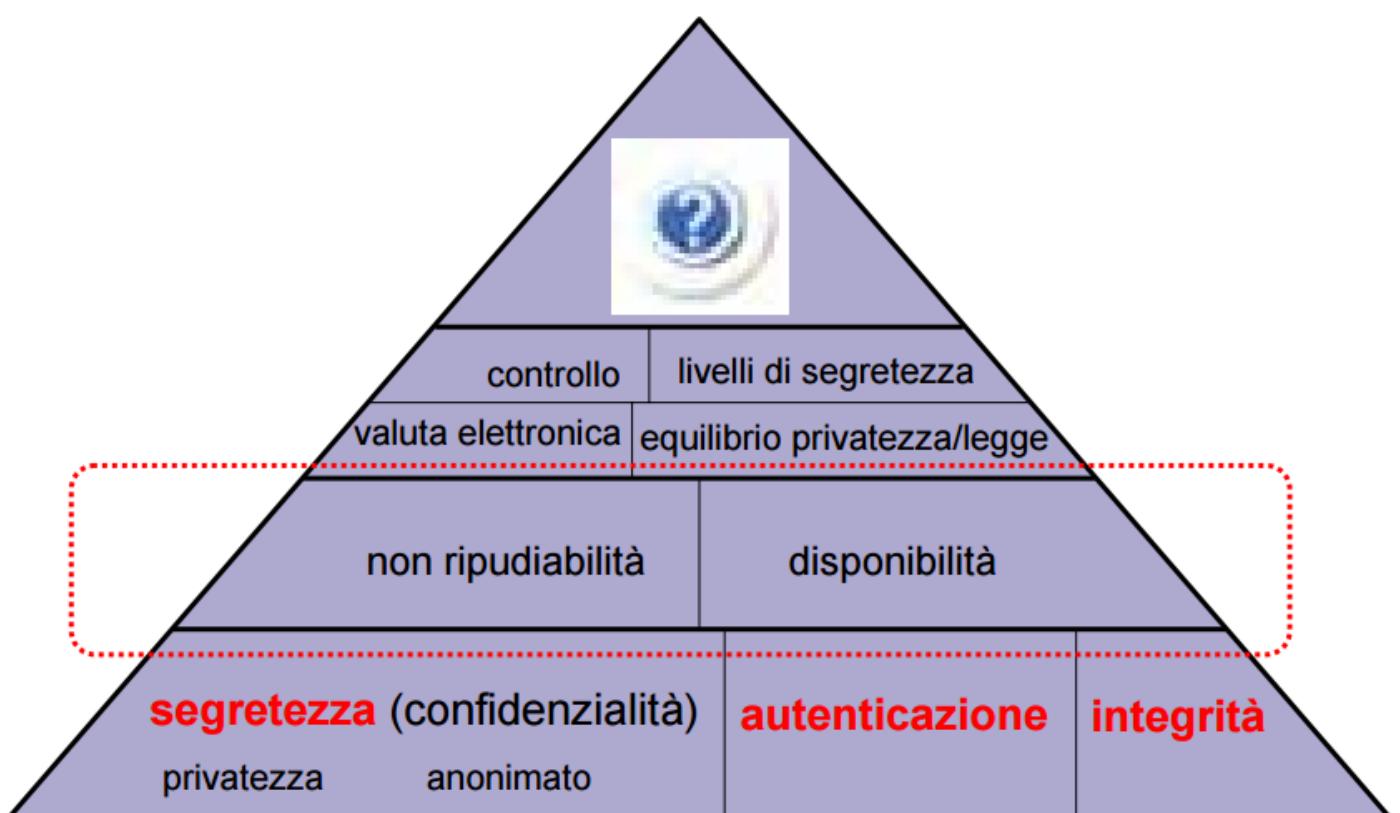
Non c'è solo Tor (opera a livello IP) che fa questo. Ci sono anche proxy che danno questo servizio. Un proxy è un programma che si interpone tra un client ed un server facendo da tramite o interfaccia tra i due host, ovvero inoltrando le

richieste e le risposte dall'uno all'altro. Il client si collega al proxy invece che al server, e gli invia delle richieste. Il proxy a sua volta si collega al server e inoltra la richiesta del client, riceve la risposta e la inoltra al client.

A differenza di bridge e router, che lavorano ad un livello ISO/OSI più basso (in quanto sfruttano i NAT), i proxy nella maggior parte dei casi lavorano a livello applicativo; di conseguenza, un programma proxy gestisce un numero limitato di protocolli applicativi.

Quindi per accedere a determinati servizi posso utilizzare altri servizi che mi rendono anonimo; proxy anonimizzatori, server anonimizzatori, dei servizi di anonimizzazione, che ci dicono sostanzialmente *"fidati di me", passa attraverso di me, e non dirò mai a nessuno quello che ti serve. Ti faccio da intermediario*. Questo è quello che fanno i server anonimizzatori. Ma questo anonimato con server anonimizzatore è uno pseudo-anonimato perché se rompiamo il proxy rompiamo l'anonimato.

## Proprietà di Livello 2



## Non-ripudio

### Definizione

L'entità non possa negare la propria partecipazione ad una transazione con uno specifico ruolo

Il non-ripudio consiste sul fatto che nessuno può negare la propria partecipazione a una transazione. È un qualcosa di rivoluzionario, perché la segretezza implica che due individui hanno un segreto condiviso tra loro. Un esempio di non ripudio è quello della compravendita; in cui uno promette di inviare la merce e l'altro di pagarla, però poi uno dei due potrebbe cercare di imbrogliare. Quindi lo scenario è diverso (tra non-ripudio e segretezza). Un altro esempio di non-ripudio è la posta elettronica certificata, perché la certificazione garantisce la spedizione della posta (pensiamo alla ricevuta di ritorno, ecc... ).

## Autenticazione, anonimato e non-ripudio

Ecco le relazioni logiche tra autenticazione, anonimato e non-ripudio:

### 1. *autenticazione* $\rightarrow$ $\neg$ *anonimato*

- Se siamo autenticati non possiamo essere anonimi. Quindi autenticazione e anonimato sono uno l'opposto dell'altro. Ad esempio quando andiamo a fare un esame il docente controlla il documento, ci vuole autenticare, non possiamo restare anonimi.
- Spesso c'è confusione tra non-ripudio e autenticazione. La differenza c'è. Il fatto che il docente controlli il documento, quindi siamo autenticati, implica che noi possiamo negare di essere venuti a fare l'esame scritto? Lo studente potrebbe dire "professore mi ha dato 18 ma io non c'ero". Il docente potrebbe dire allo studente "io ho controllato il suo documento". Come lo dimostra il docente questa cosa? In generale la non-ripudiabilità ha a che fare con la dimostrabilità a qualcun altro che ci sia stata una falsità. Il fatto che il docente controlli il documento non mi permette di dimostrare a qualcun altro che noi abbiamo fatto l'esame. Un modo per dimostrarlo è quello di fare una foto, di farci firmare. Questi sono strumenti di non-ripudiabilità.

### 2. *anonimato* $\rightarrow$ $\neg$ *autenticazione*

- La 2) è una contrappositiva della 1) (diciamo che sono la stessa cosa)
- *autenticazione*  $\equiv$   $\neg$  *anonimato*
  - Quel simbolo di equivalenza sta a indicare il "se e solo se". Questa equivalenza la si ottiene con la dimostrazione dei primi due punti.

### **3. autenticazione → non – ripudio**

- Dimostrato prima nel punto 2)

### **4. non – ripudio → autenticazione**

- Se ho partecipato a una transazione vuol dire che sono autenticato, quindi so chi è.

### **5. anonimato → ¬ non – ripudio**

- Se un utente è anonimo, come posso inchiodare una sua falsa affermazione? Cioè se l'utente è anonimo può negare la sua presenza. Per dimostrarlo posso fare così:

- La 2) dice che l'anonimato è uguale alla non autenticazione e dalla contrapposita della 4), la non autenticazione implica il non-ripudio.

### **6. non – ripudio → ¬ anonimato**

- Per dimostrarla possiamo considerare pure la 1)
- Il fatto che il docente possa dimostrare la nostra presenza all'esame, implica che noi siamo stati autenticati.

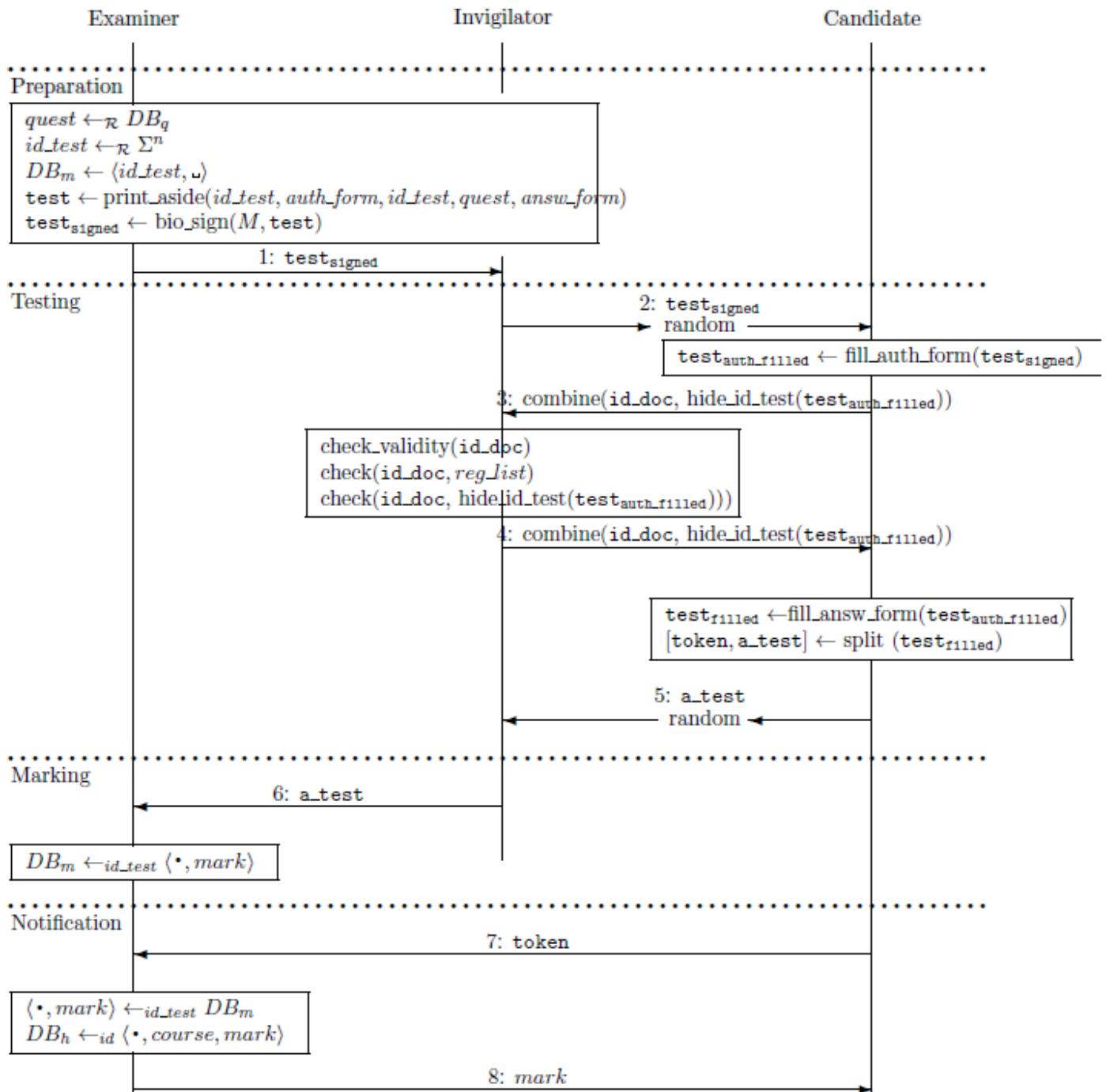
## **Esempio reale**

Si può essere autenticato e anonimo allo stesso tempo? Consideriamo l'esame scritto di sicurezza. È stato ideato un protocollo detto **WATA (Written Authenticated Though Anonymous)**. Quest'ultimo permetteva al professore di correggere i compiti scritti senza sapere di chi fosse quel compito. Quindi garantiva allo studente di essere autenticati e anonimi allo "stesso tempo".

### **Foglio d'esame di WATA2**

Name : <input style="width: 150px; height: 15px; border: none;" type="text"/>	Surname : <input style="width: 150px; height: 15px; border: none;" type="text"/>	 Cut here
ENRL Number : <input style="width: 150px; height: 15px; border: none;" type="text"/>	Date : <input style="width: 150px; height: 15px; border: none;" type="text"/>	
 Signature _____		
1) How Did He/She invent the hot water?		
<hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/> <hr/>		

Il protocollo sta scritto (in parte) qui:



Consideriamo: il candidato, l'esaminatore e il vigilante.

- **1° BOX:** l'esaminatore estrae a random da un database le domande e per questo compito verrà assegnato un identificativo casuale. Successivamente nel database verrà memorizzato l'identificativo del compito e sarà lasciato uno spazio vuoto che conterrà il voto. Nella penultima riga, viene mandato in stampa il compito (print\_aside) contenente: l'id del test, authentication form, poi nuovamente l'id del test, le domande e infine il modulo di risposta (immagine della pagina precedente). Nell'ultima riga viene costruito il test firmato. In questo caso l'esaminatore firma il token (M). Questa è una fase importante. Cioè bisogna garantire l'anonimato dello studente, e per questo motivo il docente firma prima di consegnare il compito.

- **2-4° BOX:** Vengono consegnati i compiti ai candidati in modo random. Lo studente riempirà l'authentication form con sopra la firma del docente. Il candidato prende il documento di identità e riempie l'authentication form; così quando il docente passerà, verificherà il tutto. Il docente mette la firma prima perchè, lo studente, nel momento della verifica dell'identità, può nascondere il bar-code senza farlo vedere al docente. Se il docente metteva la firma dopo, doveva necessariamente guardare pure il bar-code, e in questo caso riusciva a capire chi è lo studente di quel determinato compito (random). Il candidato ritaglia il token e può iniziare a rispondere alle domande.
- **Successivamente:** Dopo aver ritirato i compiti, essi verranno posizionati come se fossero una “pila non ordinata” (in base alla consegna). Quindi verranno posizionati in modo random (poiché se al docente gli sta antipatico uno studente e si memorizza “la posizione” del compito di quel studente allora non abbiamo concluso gran che. Per questo motivo vengono posizionati random i compiti alla consegna)
- **5° BOX:** Dopo aver corretto e stabilito il voto del compito, viene inserito nel database il voto associato a quel determinato compito. Quindi, nel database, questa tabella conterrà le voci [id\_compolto, voto].
- Infine lo studente si presenta con il token e scoprirà il voto del compito e successivamente gli veniva registrata la materia. Ovviamente il docente aveva anche una tabella con gli studenti che dovevano fare quel compito, così alla fine sapeva a chi registrare quel voto.

Con questo procedimento non è impossibile avere utenti autenticati e anonimati “contemporaneamente”. Ma **attenzione**, non accade nello stesso istante, cioè prima l’utente si autentica, ma solo in un secondo momento diventa anonimato. Questo passaggio, avveniva nel momento in cui viene rimosso il token dal compito.

## Disponibilità

### Definizione

Il sistema sia operante e funzionante in ogni momento

La disponibilità la vorremmo tanto nei nostri servizi elettronici, in particolare, ad esempio, nessuno vorrebbe che il proprio sito bancario fosse indisponibile. L’associazione tra mancata disponibilità di un sito, di un servizio e la sua generale inaffidabilità, che non va bene ecc.. è un’associazione sulla quale lavorano i sociologi.

La disponibilità la possiamo considerare tra i requisiti funzionali e quelli non funzionali; perché uno vorrebbe che il sito funzioni e questo deve funzionare in maniera affidabile, quindi non ci deve essere dietro un attaccante che mi fa “danni”.

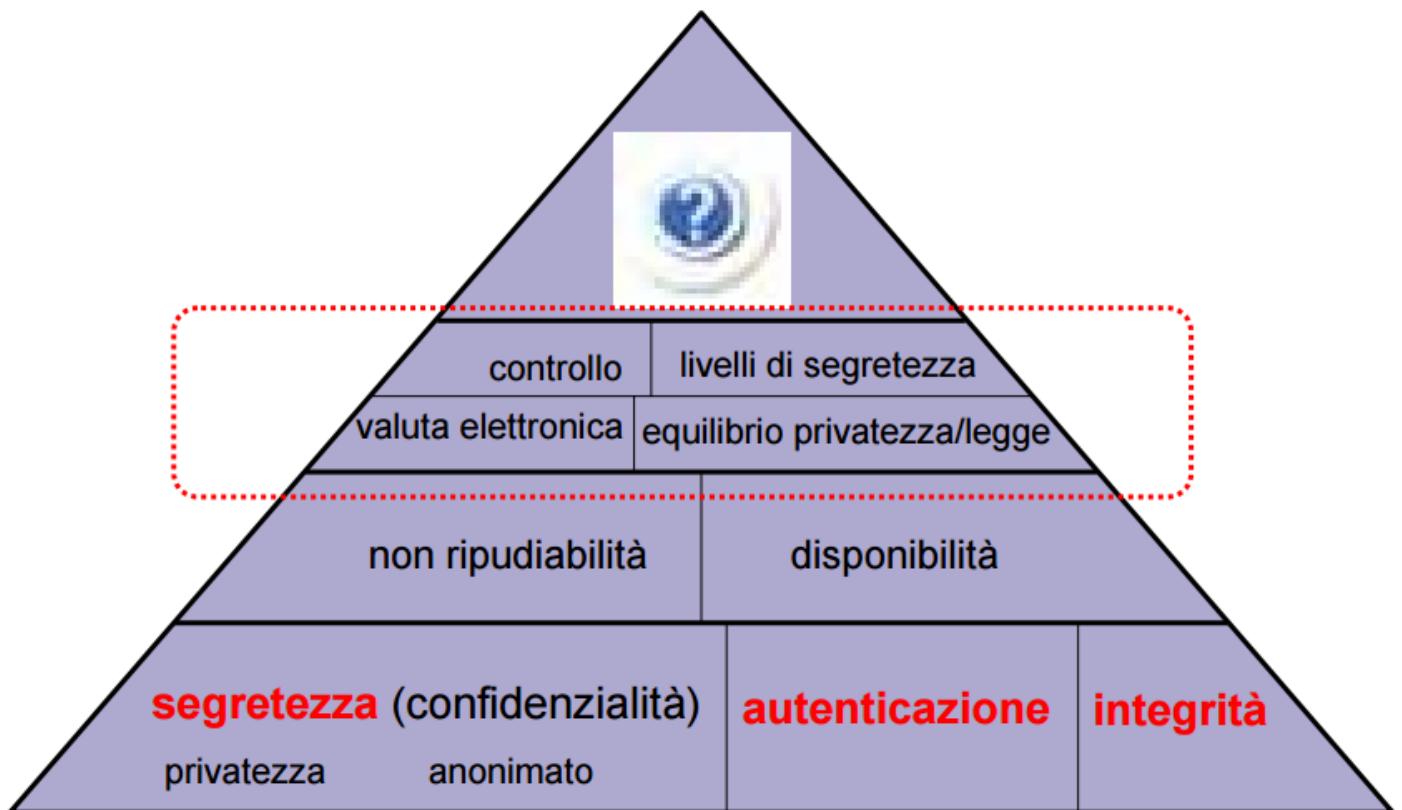
## Alcune misure

- Limitare accesso ad utenti autenticati (?)
- Complicare accesso al sistema impegnando computazionalmente il chiamante
- Sostanzialmente garantire la disponibilità rimane un problema aperto

Abbiamo il problema tra la disponibilità e la negazione del servizio. Un client potrebbe mandare tante richieste a un servizio online (**flooding di richieste**). Questo il client lo può fare, ma come deve reagire il sito? Il sito, avrà tante richieste http. Come potrebbe risolvere questo problema? Se chiude la porta non possiamo fare più niente, quindi non va bene. Ci vorrebbe la capacità di rispondere "a scatti". Con queste euristiche (un tentativo di soluzione che non funziona in generale) qualcosa di buono si può fare. In particolare ci sono applicazioni di basso livello che permettono di scrutinare il traffico in ingresso a intervalli. Significa che ciò che c'è a un certo intervallo di chiusura temporanea viene eliminato. È come se rispondessimo a una domanda su cinque (dove queste cinque vengono fatte contemporaneamente).

Un altro protocollo (per cercare di risolvere il problema descritto prima), si chiama **cookie transformation**, riguarda di complicare l'accesso al sistema impegnando computazionalmente il chiamante. Questo protocollo funziona così: noi facciamo una domanda, il destinatario prima di rispondere ci fa un'altra domanda. Quindi se il chiamante facesse un ciclo infinito di flooding, allora anche lui deve impegnarsi computazionalmente. Rispondere con un'altra domanda è computazionalmente meno impegnativo di rispondere a una risposta voluta (rispondere a una domanda con un'altra domanda ti impegna meno che dare la risposta a quella domanda).

## Proprietà di livello 3



Al livello tre stiamo considerando qualcosa di molto applicativo. Per controllo in generale riguarda il controllo di accesso, controllo di legittimità. Dei livelli di segretezza riguardano sistemi multilivello, tipicamente militari.

### Controllo di accesso

#### Definizione

Ciascun utente abbia accesso a tutte e sole le risorse o i servizi per i quali è autorizzato

Il controllo di accesso lo possiamo considerare come istanza della proprietà di controllo sicuro.

Consideriamo l'esempio di trojan (presente nelle prime pagine di questo documento). Esso è un modo per bypassare il controllo di accesso, perché l'utente **giamp** riusciva a spazzar via lo spazio dell'utente vittima. Quindi come esempio di controllo di accesso possiamo considerare che due utenti della stessa macchina non si possano danneggiare a vicenda.

#### Esempi

*Accesso fisico ad aree di un edificio, accesso digitale a spazio disco, etc.*

Ecco alcune misure per il controllo di accesso:

- **Autenticazione dell'utente:** dobbiamo controllare ogni utente; che cosa possa fare, che cosa deve fare. Quindi prima di tutto dobbiamo controllare che tipo di utente sia.
- **Politiche di sicurezza.**
- **Implementazione delle politiche:** ad esempio le ACL in Linux.

## Esempio di politica di sicurezza

Il problema delle politiche di sicurezza sono le regole che dobbiamo dare. Senza di esse non ci possiamo aspettare che il sistema sia sicuro.

- **Ex1 di politica:** L'utente A del mio sistema Ubuntu non possa accedere all'applicativo password per il cambio delle password.
- **Ex2 di politica:**
  1. Un utente ha il permesso di leggere un qualunque file pubblico
  2. Un utente ha il permesso di scrivere solo sui file pubblici di sua proprietà
  3. Un utente ha il divieto di sostituire un file con una sua versione più obsoleta
  4. Un utente ha l'obbligo di cambiare la propria password quando questa scade
  5. Un utente segreto ha il permesso di leggere su un qualunque file non pubblico
  6. Un utente segreto ha il permesso di scrivere su un qualunque file non pubblico
  7. Un amministratore ha il permesso di sostituire un qualunque file con una sua versione più obsoleta
  8. Un utente che non cambia la sua password scaduta (negligente) ha il divieto di compiere qualunque operazione
  9. Un utente che non cambia la sua password scaduta (negligente) non ha discrezione di cambiarla

Le singole regole sono perfette. Il problema è la convivenza di tutte queste regole. Il problema della politica specificata prima è che dietro ci sono un sacco di inconsistenze.

## Elementi di una politica

Gli elementi di una politica di sicurezza sono:

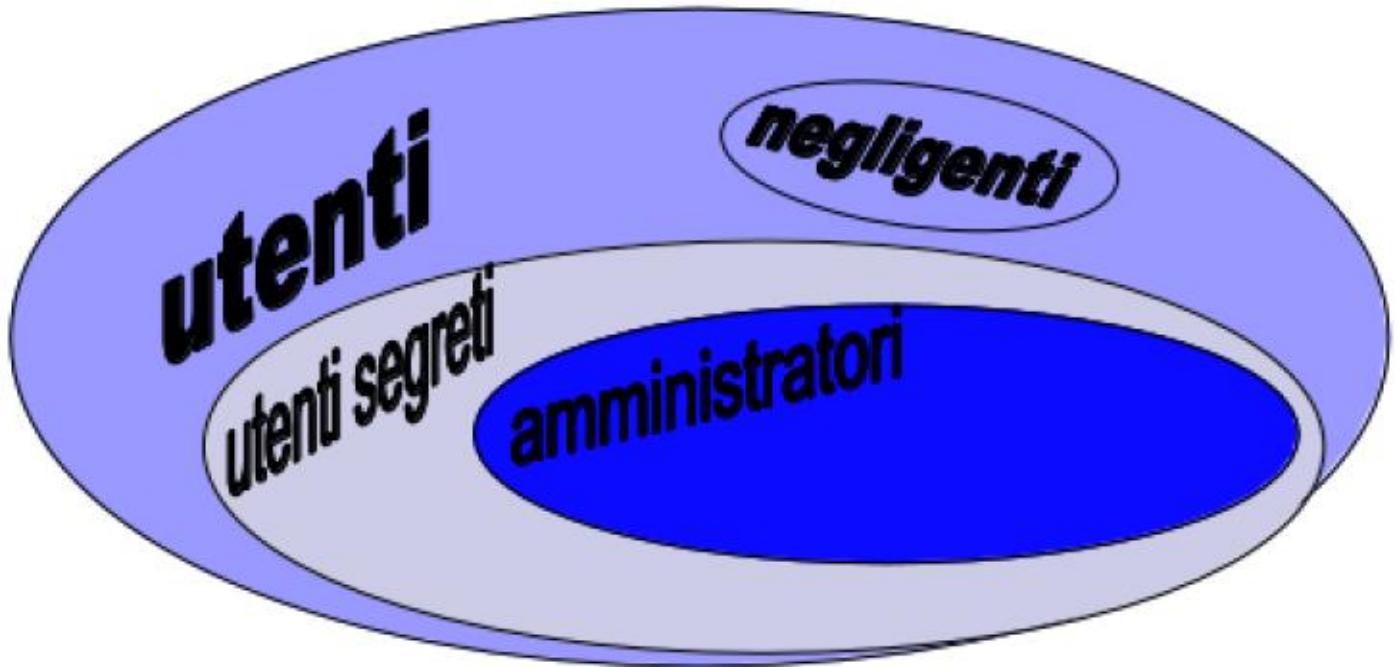
- **Ruoli**: il problema forse nasce proprio dai ruoli. E se non mettessimo i ruoli? Ad esempio consideriamo un S.O. senza dei ruoli. Ogni nuovo utente che viene registrato che politiche di accesso diamo? Possiamo stabilire che se questi sono utenti, ad esempio studenti, diamo un certo insieme di privilegi basso. Questi sono ruoli! In questo modo possiamo andare a creare un insieme di utenti che hanno gli stessi ruoli. Quindi i ruoli esistono.
- **Utente**: qualunque entità che ricopre un certo ruolo.
- **Operazioni**: leggere, scrivere, cambio password.
- **Modalità**: obbligo, permesso, divieto e discrezionalità.

Definiamo le modalità: **obbligo, permesso, divieto e discrezionalità**.

Fissiamo come **concetto base l'obbligo**. Da questo deriviamo gli altri.

- **Vietato( $x$ ) = Obbligatorio( $\neg x$ )**
  - Il divieto lo possiamo definire come l'obbligo di non fare una determinata cosa.
    - Ad esempio il divieto di fumare è l'obbligo di non fumare.
- **Permesso( $x$ ) =  $\neg$  Obbligatorio( $\neg x$ )**
  - Il permesso di fare qualcosa lo possiamo definire come il non c'è l'obbligo di non farla.
    - Ad esempio abbiamo il permesso di andare via, quindi non siamo obbligati a non andar via.
- **Discrezionale( $x$ ) =  $\neg$  Obbligatorio( $x$ )**
  - La discrezionalità di fare qualcosa la possiamo definire come il non c'è l'obbligo di fare quella cosa.

## Intersezione dei ruoli



I ruoli non sono un artificio. Esistono e senza di essi non potremmo fare nulla. Dobbiamo anche considerare la loro intersezione, perché altrimenti una particolare funzionalità, come ad esempio la possibilità di leggere o scrivere da un file, la dovremmo “copiare” ogni volta per i tipi di utenti differenti.

Il sottoinsieme più piccolo è rappresentato dagli amministratori, poi possiamo considerare gli “utenti segreti”, e infine gli utenti; ci sono anche quelli negligenti che sono quelli che non cambiano la password. Se consideriamo l’immagine di sopra, amministratori negligenti non ce ne possono essere.

## Inconsistenze di una politica

I nostri problemi di inconsistenze sono questi:

- **Contraddizione**  $\equiv \text{Obbligatorio}(x) \wedge \neg \text{Obbligatorio}(x)$ 
  - È vera una cosa e anche il suo opposto.
- **Dilemma**  $\equiv \text{Obbligatorio}(x) \wedge \text{Obbligatorio}(\neg x)$ 
  - Dilemma sul parametro. Sul parametro ci può essere un po' di ambiguità. Cioè ad esempio: è obbligatorio venire a lezione ed è obbligatorio non venire a lezione ci mette in dilemma.

Gran parte dei problemi di sicurezza basate sistemi operativi, non riguardavano solo errori di implementazione, ma questi ultimi due punti; implementazione di qualcosa risultante alla fine inconsistente.

Ex. **Role based access control** (descritto un paio di argomenti dopo)

Altri esempi di contraddizioni e dilemmi (cercare altri esempi):

- Contraddizione da regole 3 e 7
  - Un amministratore ha permesso e divieto di fare downgrade di un file
- Dilemma da regole 8 e 9
  - Un utente negligente ha l'obbligo sia di cambiare sia di non cambiare la propria password

## Politiche in pratica: MAC

Il **MAC** (**Mandatory Access Control**, controllo di accesso mandatorio) realizza politiche basate sull'obbligo. È usato in ambito militare.

**Mandatory Access Control** ( MAC ) è un meccanismo di sicurezza che limita il livello di controllo che gli utenti (soggetti) hanno sugli oggetti che essi creano. A differenza di un DAC di attuazione, in cui gli utenti hanno il pieno controllo sui propri file, directory, ecc, MAC aggiunge etichette aggiuntive, o categorie, a tutti gli oggetti del file system. Utenti e processi devono avere l'accesso appropriato a queste categorie prima di poter interagire con questi oggetti.

Sistemi operativi di alta sicurezza sono SELinux, AppArmor, Grsecurity, Tomoyo. Questi sono S.O. basati sull'obbligo e di alta sicurezza. Nel laboratorio eravamo riusciti a violare a password in Linux. Se avessimo usato SELinux non sarebbe stata la stessa cosa.

Linux, windows, ecc.. sono non mandatori. Ci sono politiche non mandatorie, ovvero, a differenza di quanto parlato prima, non sono basate sull'obbligo. Le ACL di Linux mica hanno il mio obbligo; hanno modalità più semplice, ma sempre c'è il problema di inconsistenza.

## Politiche in pratica: RBAC

**Role-Based Access Control** (RBAC), realizza politiche non mandatorie, ovvero

- basate sui permessi associati a ciascun ruolo
- modificabili

Semplificazione delle modalità

- Eliminato Obbligatorio, scompare anche Discrezionale
- Dalle definizioni di Permesso e Divieto segue:
  - $\text{Permesso}(x) = \neg \text{Vietato}(x)$
  - $\text{Vietato}(x) = \neg \text{Permesso}(x)$

Usato per sistemi operativi comuni.

## Meccanismi implementativi: ACM

Le politiche esistono. Ecco come vengono implementate.

Fissate le politiche con tutti i problemi che ci sono, abbiamo bisogno di implementarle.

Le ACM sono le matrici di controllo di accesso. Esse rappresentano lo stato dei permessi. In questa matrice possiamo specificare tutto ciò che può fare *chi* su *che* cosa, dove *chi* rappresenta i ruoli, *che cosa* rappresenta gli oggetti, i file di sistema.

Ex:

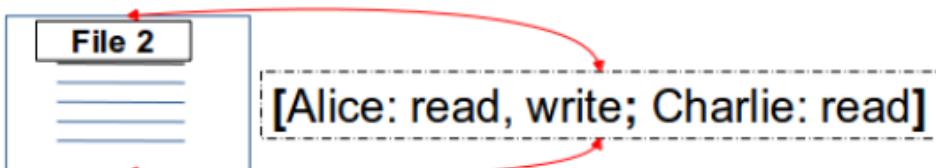
	File1	File2	File3	Program1
Alice	Read, write	Read		Execute
Bob	Read		Read, write	
Charlie		Read		Read, execute

Questa matrice è sottesa la modalità permesso, il che significa che questa matrice definisce permessi, questa matrice definisce tutte le istanza della modalità permesso sul nostro sistema. Questa matrice ci dice cosa quei ruoli possono fare su quelle risorse. In S.O. abbiamo visto le ACL. Se leggo una colonna di questa tabella ottengo una ACL. Quindi una ACL è una colonna e una CaL (Capability List) è una riga (ovviamente scrivendo la matrice al contrario le ACL sarebbero state le righe e le CaL le colonne).

## Meccanismi implementativi: ACL e CaL

### ■ ACL (access control list)

- **Definizione:** una lista di controllo degli accessi (access control list, abbreviato in ACL) è un meccanismo usato per esprimere regole complesse che determinano l'accesso o meno ad alcune risorse di un sistema informatico da parte dei suoi utenti.
- Ogni colonna dell'ACM registrata con lo specifico oggetto:



### ■ CaL (Capability List)

- **Idea:** associare *ad ogni soggetto* la lista degli oggetti ai quali ha accesso insieme all'elenco delle operazioni consentite per ciascuno di essi
- Nella rappresentazione interna di un soggetto esiste la lista delle capability del soggetto
- Se il soggetto X può invocare l'operazione op sull'oggetto Y allora X ha una **capability(Y,op)**

- Ad ogni invocazione di un'operazione si controlla che esista una capability che lo permetta
- Il controllo avviene nell'ambiente del soggetto
- Ogni riga dell'ACM registrata con lo specifico soggetto



## Tassonomia di attaccanti

(Approfondire dal libro)

- Vari moventi
  - Ricchezza
  - Informazioni sensibili
  - Potere
  - Gloria
  - Divertimento
  - :

- Varie classificazioni
  - Diritti
  - Risorse
  - Esperienza
  - Rischio accettato
  - :

1. Hacker (cracker),
2. Attaccanti interni,
3. Spionaggio industriale,
4. Servizi segreti,
5. Organizzazioni criminali/terroristiche,
6. Difesa

## Modelli di attaccanti

### Definizione

Un modello di attaccante specifica (le capacità offensive di) un preciso attaccante

I modelli di attaccante ci consentono di astrarre dal contesto; ci permettono di astrarre dal fatto che si tratti di spionaggi industriali, ecc.. (*un po' come la complessità asintotica ci permette di astrarre dalle capacità computazionali della macchina*).

**Un modello di attaccante specifica le capacità offensive di un preciso attaccante.** Quindi stiamo astraendo dal preciso individuo, realtà sociale, contesto, ecc... Stiamo definendo un insieme di capacità offensive. Fissate un insieme di capacità offensive, gli studi di sicurezza li possiamo fare in prelazione a quelle capacità offensive. Fissato il modello, diremo: questo sistema è sicuro nei confronti di quel set che noi chiamiamo modello di attaccante di riferimento? Un modello di attaccante notevole si chiama **Dolev-Yao** (dai nomi degli inventori, in data 1983). Questo modello sarà un modello “pessimo”, ovvero il peggiore possibile in termini di capacità offensiva (quindi è un modello massimamente offensivo). Questo modello è realistico nella misura in cui, se noi riusciamo a dimostrare che un sistema è infallibile nei confronti di un attaccante peggiore immaginabile, allora poi tutto quello che ci siamo dimenticati nel contesto reale non fanno gioco, nel senso che se abbiamo dimostrato che il sistema è sicuro nei confronti dell'attaccante massimamente offensivo, allora a maggior ragione sarà sicuro con un attaccante meno offensivo (questa è l'argomentazione tipica che giustifica un modello di attaccante super potente). Quindi in generale se una macchina è sicura nei confronti di un bambino dell'asilo, magari non è sicura nei confronti di uno studente di Internet Security. Dobbiamo sempre specificare se un sistema è sicuro su che cosa è sicuro. Se questo “che cosa” lo definiamo come massimamente offensivo, allora se il sistema è sicuro da questo modello, a maggior ragione sarà sicuro da altre capacità offensive che posso immaginare.

**Domanda:** Ma quanto mi garantisce che questo modello è massimamente offensivo?

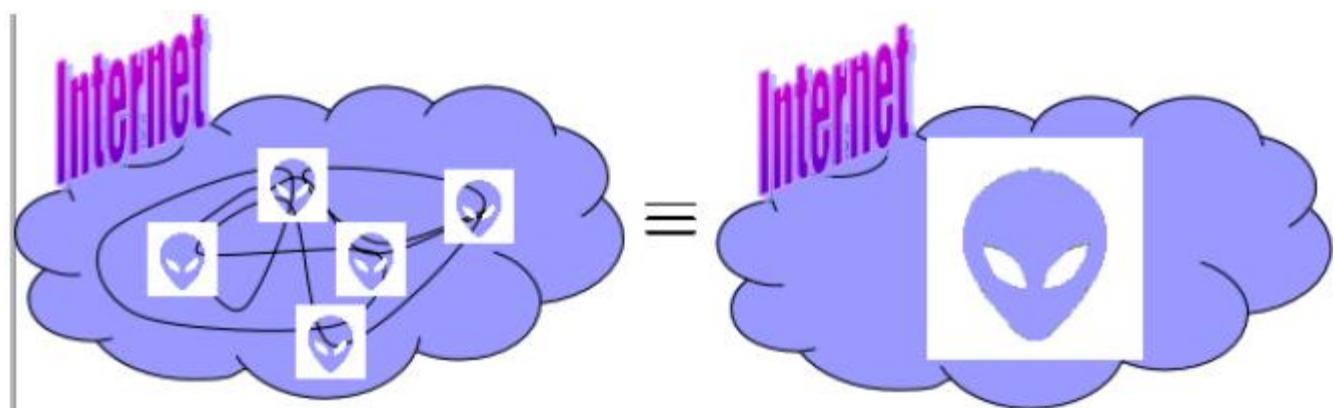
**Risposta:** non importa, perché il nostro scopo è riuscire a dimostrare che il sistema è sicuro nei confronti di questo modello terribile, così che poi sarà sicuro in pratica. Quindi non importa che riusciamo ad incorporare un modello il più terribile possibile; ma la cosa importante è che il sistema sarà sicuro nei confronti del modello più offensivo possibile.

## Modello di attaccante Dolev-Yao (DY)

### Definizione

L'attaccante è unico e superpotente, ovvero controlla l'intera rete, ma non può violare la crittografia

Nel 1983 non c'erano problemi di sicurezza come oggi; non c'era il problema di sicurezza del bit-coin; non c'era lo sniffer. C'erano pochi enti universitari, pochi enti governativi connessi alla rete. Quindi il contesto di sicurezza dell'epoca era quello di "due spie" che volessero comunicare in maniera sicura pur trovandosi dislogati. Cioè queste due entità si fidano l'una dell'altra, ma che non si fidano del resto del mondo. Quindi il loro obiettivo era quello di stabilire un canale di comunicazione sicuro per impedire che tutto il resto del mondo possa scoprire, intercettare le loro comunicazioni, o peggio ancora spacciarsi per uno di loro. Questo scenario si applica al non ripudio. Quindi il modello di attaccante è questo: l'attaccante sta in mezzo ai due interlocutori che si fidano l'uno dell'altro, il quale può fare sostanzialmente tutto. Ci sono degli studi di equivalenza che dimostrano che se abbiamo due attaccanti con un certo livello di capacità offensive si possono dimostrare equivalenti termini di capacità offensive al singolo attaccante DY. Si è dimostrato che il modello DY è il più potente di tutte le distribuzioni di capacità offensive che possiamo pensare.



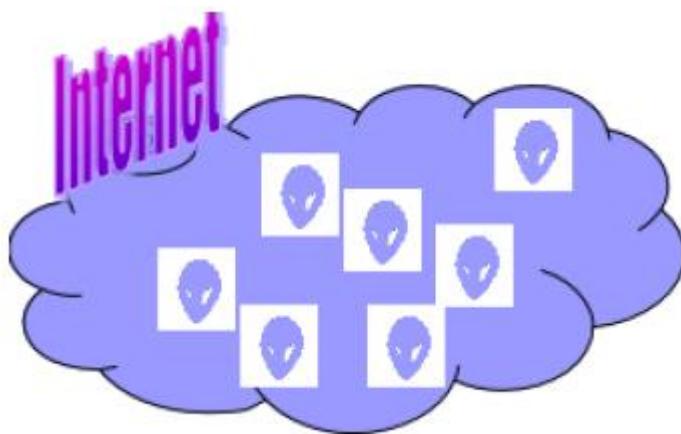
Il disegno fa significare che sulla rete c'è un unico attaccante super potente (il discorso fatto prima).

## Modello di attaccante General Attacker (GA)

### Definizione

Chiunque può essere attaccante senza interesse a colludere con altri, al peggio con capacità di totale controllo della rete, ma senza violare la crittografia

DY non è più aderente al panorama attuale, perché al giorno d'oggi la rete, le capacità offensive ce li abbiamo ognuno di noi. Quindi poiché il contesto socio tecnologico è cambiato, consideriamo attaccanti dislogati.



Prima però è stato detto che erano “equivalenti” gli attaccanti; ora non lo sono più, perché il contesto socio tecnologico è cambiato, perché ciascuno di noi abbiano capacità offensive. Quindi lo scenario di condivisione degli obiettivi da parte di attaccanti dislogati, rappresentata da DY negli anni '70, oggi non è più tanto reale, perché due utenti potrebbero non voler colludere nei confronti di un certo obiettivo di violazione, perché questi due utenti potrebbero avere lo stesso interesse, non condiviso, nei confronti della violazione di una transazione ebay (per esempio). Quindi lo scenario di equivalenza non si applica più. Il teorema di equivalenza descritto prima diceva che: se ci sono due attaccanti con lo stesso obiettivo, che vogliono colludere, togliamoli e immaginiamo che ci sia DY; ma devono **voler colludere**. Quindi questi “attaccanti generali” non hanno interessi a colludere, ma possono avere interessi del tutto personali.

# **AUTENTICAZIONE**

## Varie ambientazioni

Esistono vari tipi di autenticazione:

- **Persona-persona:** ogni volta che ci si incontra
- **Utente-computer:** ogni volta che vogliamo fare login su una macchina multiutente
- **Computer-computer:** detta spesso punto-punto
- **Computer-utente:** è il computer che ci chiede la password; ma non autentichiamo mai il computer? Il computer al giorno d'oggi è un insieme di servizi, quindi ci poniamo il problema se stiamo mettendo il numero di carta di credito sul sito dell'Alitalia o su un altro sito.
- **Utente-utente:** è più rara, ma si può fare.

Spesso sono richieste varie combinazioni di queste autenticazioni, affinché l'utente possa usufruire di un servizio. Quindi come minimo ci sarà l'autenticazione utente-computer; ci potrebbe essere la computer-utente; ci potrebbe essere un "tunnel" sicuro e quindi l'autenticazione computer-computer. Tutto ciò è trasparente per l'utente, ma in realtà esiste.

L'autenticazione si fa per mezzo di qualcosa che ci caratterizzi, perché per la definizione di autenticazione ognuno è quello che dice di essere. Quindi se ci arriva una email da Giampaolo Bella questa email tipicamente non sarà autenticata, ma se mi arriva una email della caratterizzazione di Giampaolo Bella allora discerneremo che è veramente Giampaolo Bella. I modi per discernere sono:

- **Conoscenza**
- **Possesso**
- **Informazioni biometriche** (La firma a penna è una firma biometrica, tradizionalmente usata per autenticare un documento).

Possiamo porci la seguente domanda:

- **Qual è la tecnica più robusta di autenticazione? Quella basata su possesso e conoscenza?**

È un concetto semplice per la nostra epoca, perché ad esempio se il bancomat non avesse il pin, quindi sarebbe solo autenticazione basata sul possesso, sarebbe poco robusta. Quindi dobbiamo fare autenticazione basata sul possesso, (con la carta) e sulla conoscenza (ovvero avere il pin). Quindi sia se considero solo conoscenza o solo possesso, queste tecniche sono "deboli" da sole; ma si possono potenziare l'una con l'altra.

# Autenticazione basata su conoscenza

Conoscere la giusta password comprova identità.

Rischi che possiamo correre (sulla password):

- **Guessing:** password indovinata (attacco standard, attacco dizionario, attacco forzabruta)
- **Snooping:** sbirciata mentre viene inserita
- **Spoofing:** scoperta tramite falsa interfaccia di login (trojan)
- **Sniffing:** intercettata durante la trasmissione

Osservazione basilare: la conoscenza di una password ci apre un mondo di servizi riservati a quella password.

## Guessing - attacchi

*Come si indovina una password?*

### 1 Attacco standard

- Password brevi, tipiche, relative all'utente (*hobby, nomi parenti, compleanno, indirizzi*)

### 2 Attacco dizionario

- Vengono provate tutte le parole di un dizionario
- Tentativi spesso arricchiti con regole che ricalcano possibili scelte dell'utente (*doppia parola, parola al contrario, 0 al posto di o, 1 al posto di i*)

### 3 Attacco forza bruta

- Vengono provate esaustivamente tutte le parole costruibili in un dato vocabolario (*alfanumerico, simboli speciali*) di lunghezza via via crescente (1, 2, ..., 6, ...)
- Empiricamente si vede che con un ricco vocabolario, 6 è una soglia notevole

Per scoprire una password, abbiamo questi tipi di attacchi:

- **Attacchi standard:** (vedere figura) ad esempio accediamo su account con password banali: utente = *user*, password = *user* (ne esistono tanti account di questo tipo).
- **Attacchi dizionario:** molto spesso vengono usate come password parole presenti in dizionari (di qualsiasi lingua). Per evitare di usare password di questo tipo, solitamente, vengono inseriti numeri e altri caratteri nella parola. Quindi usando come password parole di dizionari, in una notte è possibile sfogliare dizionari italiani, inglesi,... e trovare la password. Alcune volte

vengono usate dei piccoli trucchetti: ad esempio la ‘o’ (o-minuscola) diventa ‘O’ (O-maiuscola), le ‘i’ diventano ‘1’, ecc...

- **Attacco forza bruta:** ci sono password veramente complicate, perché sono combinati caratteri speciali, ecc... Si può calcolare la probabilità per cui possiamo indovinare una password. Perché non è detto che se la password sia lunga 10 caratteri è una password che non può essere indovinata. Tale probabilità è (1 diviso la cardinalità dell’alfabeto elevato alla lunghezza della password)

$$Pr[\text{guess}(n)] = \frac{1}{|A|^n}$$

Quindi se ad esempio abbiamo una password binaria, allora la probabilità di indovinarla è  $1/2^n$ . Più è grande il denominatore e più la probabilità di indovinare la password è minore.

Se non conoscessimo la lunghezza della password, in modo esaustivo, potremmo provare tutti i possibili valori di n (quindi prima vedremo tutte le possibili password per n=2, poi per n=3, ecc...).

## Guessing - contromisure

### 1 Controllo sulla password

- Il sistema controlla che la password non sia banale (*lunghezza, caratteri speciali*) quando essa viene scelta

### 2 Controllo sul numero inserimenti

- Il sistema limita i tentativi di login, pena blocco

### 3 Uso di CAPTCHA

- Completely Automated Public Turing Test To Tell Computers and Humans Apart
- Il sistema richiede la risoluzione di una captcha per verificare che il tentativo di login viene da un umano
- Recente algoritmo basato su Google Street View viola 99% delle captcha alfanumeriche — @nassecuritynews

Le contromisure che vengono per questi tipi di attacchi sono:

- Il sistema controlla che la password non sia banale;
- Il sistema limita i tentativi di login, pena il blocco. Ad esempio supponiamo di violare un servizio di rete; quindi possiamo automatizzare delle query per cercare di violare il servizio. Ovviamente ci auguriamo che il servizio di rete butti fuori l’attaccante che prova ad effettuare tutti questi tentativi.

- Uso di **CAPTCHA**. Contro l'attacco forza bruta (provare esaustivamente tutte le parole possibili di un vocabolario), non esiste solo la tecnica di limitare i tentativi di accesso, ma si usano anche i **captcha**. Esse vengono usate per dimostrare che non siamo robot. Al giorno d'oggi le tecniche di computer vision sono molto avanzate (*vision = riconoscere informazioni salienti da un'immagine*). C'è un recente algoritmo basato su **Google Street View** che viola il 99% delle captcha alfanumerici. Quindi il 99% delle captcha alfanumerici sono violabili da un robot. Google ha dovuto risolvere questo problema, quando ha cominciato a filmare tutte le strade, perché su di esse ci sono i numeri civici. Allora Google ha avuto il problema di riconoscere i numeri civici. Se noi vogliamo una captcha sul nostro servizio basta chiedere a Google (ha comprato una start-up che faceva questo lavoro). Molte delle captcha che vengono fuori servono numeri civici. Quindi circa nel 2011-12, Google ebbe il problema di risolvere i numeri civici che firmavano Google Street. Risolvere nel senso trasformare l'immagine in informazioni digitali. Qualcuno (il tutto non è stato confermato) pensò di fare questo: se all'utente spunta un captcha con i caratteri 1 e 8, allora esso scriverà 18 e conserveremo questa informazione in un database; così associamo a quel captcha che corrisponde il numero 18. Se l'utente scriveva un numero sbagliato questo veniva salvato (quindi quel captcha corrispondeva un numero sbagliato). Questa tecnica non va bene. Successivamente è stato pubblicato da un gruppo di ricercatori un articolo con un algoritmo che riconosceva il 99% delle captcha automaticamente. Quindi il problema è stato risolto; è bastato costruire il robot che risolveva il problema. (Da qualche parte c'è scritto che: “*Google dice che riesce a riconoscere se è un robot e no un umano che sta eseguendo quel servizio senza aver bisogno di captcha!*”).

## Norme per una buona password

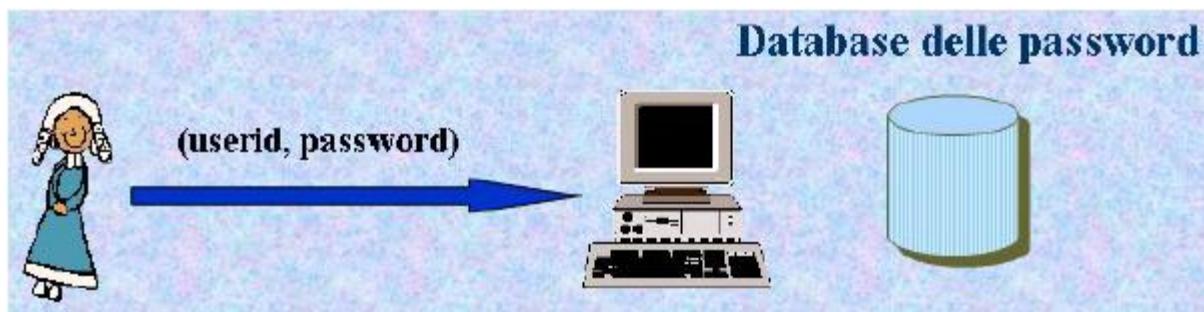
### Norma fondamentale

Bilanciare al meglio **mnemonicità** (sia facile da ricordare cosicchè non serve scriverla) e **complessità** (sia robusta verso guessing)

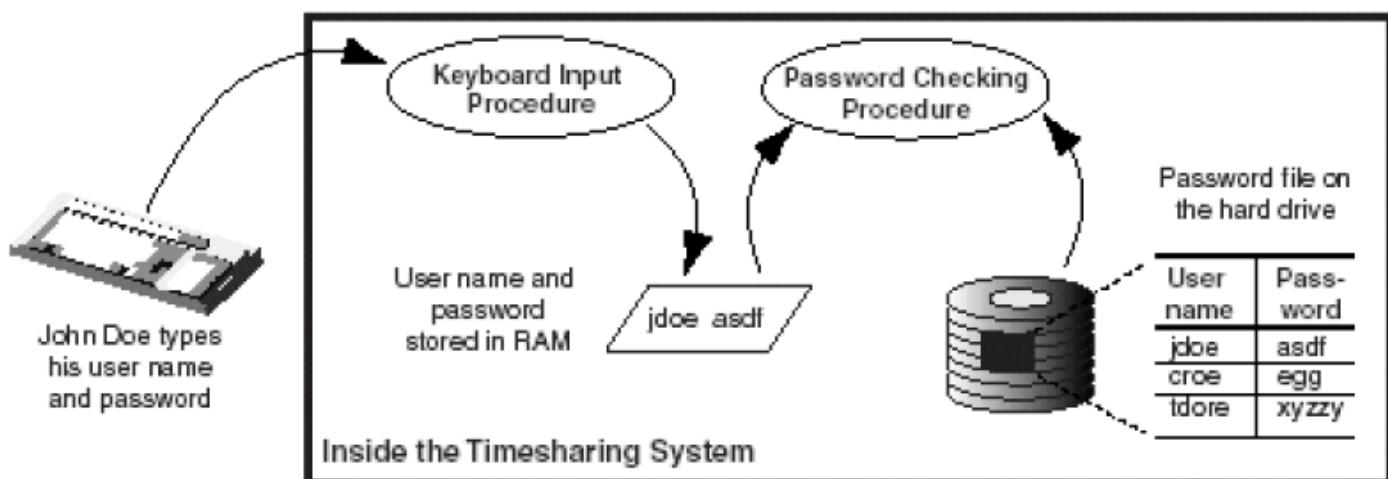
Il problema fondamentale è che vorremmo usare una password, ad esempio, di 25 caratteri per ogni servizio, così sarà molto difficile da farla indovinare a un attaccante. Però se l'utente usa molti servizi viene molto complicato ricordare tutte queste password. Quindi il problema fondamentale dell'uso di una buona password è il delicato rapporto tra **mnemonicità** (cioè di ricordabilità) e **complessità** (cioè difficoltà nell'indovinarla).

## Mantenere una password & Storia: CTSS

La password va mantenuta in qualche modo nel sistema, perché va riconosciuta. La cerimonia è ovvia: un utente inserisce la password, si fa il matching e se questa operazione va bene possiamo accedere al sistema.



Quindi la password è conservata in un database in una qualche forma. Al giorno d'oggi la password non è memorizzata in chiaro. Storicamente la password veniva conservata in chiaro, perché c'erano le politiche di sicurezza, c'era il controllo di accesso. C'è una circolarità dell'idea: consideriamo il controllo di accesso; ci serve una politica di sicurezza, un sistema di autenticazione e un meccanismo implementativo per la politica, ovvero per il controllo di accesso. Quindi il controllo di accesso dipende dall'autenticazione e quindi (ora si sta dicendo) che l'autenticazione dipende dal controllo di accesso, perché metto la password in chiaro e posso dire che sono protetto dal controllo di accesso. Ma questo al giorno d'oggi non può funzionare! (cioè possiamo conservare la password in chiaro).



L'immagine sopra racconta proprio la cerimonia. L'utente inserisce *user name* e *password* attraverso la tastiera (Ex. Utente: jdoe, password: asdf), c'è una procedura di controllo che sa la *password live* e la *password registrata* nella tabella. Il database era composto da una semplice tabella con campi User name e Password. Al giorno d'oggi una tabella del genere ci fa ridere (**ma 50 anni fa no**).

## Autenticazione basata su possesso

L'obiettivo è sempre lo stesso; convalidare l'identità dell'utente: comprovata dal possesso del giusto oggetto, tipicamente magnetico o elettronico (carta magnetica, smart card, smart token).

L'oggetto può memorizzare informazione sensibile

- Informazione su carta magnetica interamente leggibile
- Informazione su carta elettronica leggibile coerentemente con interfaccia funzionale

Le smart card sono abbastanza potenti. Le smart token sono sempre più diffusi per autenticazioni sensibili (banca online, ...). Uno smart token, a differenza di una smart card, ha un interfaccia di I/O utente (la smart card ha I/O macchina). Lo smart token può avere un pulsante o una tastiera per immettere il pin (la differenza con le smart card è sostanzialmente questa).

**Perché non bastano le smart card?** Il processo di autenticazione riconosce l'oggetto e no l'utente. Quindi se facciamo autenticazione con possesso e inseriamo la carta stiamo riconoscendo la carta e no l'utente (come nel caso della conoscenza; cioè a causa della conoscenza riconoscevamo l'utente).



Pertanto l'autenticazione si fa a due vie: si combina l'autenticazione basata sulla conoscenza e l'autenticazione basata sul possesso. Questo per il bancomat è semplice. Con i siti web come si combinano queste due cose? Uno smart token è un mini computer. L'utente si deve autenticare ad esso (non sempre). Quindi l'utente deve solo premere il bottone che sta sullo smart token per autenticarsi. L'utilizzo di quello smart token è al portatore. Genera uno speciale segreto detto **one-time password**, accettato solo una o poche volte dal server. Il server riconosce questo segreto poiché essi sono sincronizzati computazionalmente; cioè eseguono lo stesso algoritmo a partire dallo stesso seme con entropia esterna (tipicamente il tempo corrente), e quindi sono sincronizzati computazionalmente. L'autenticazione al server web, oggi è basata sul possesso e sulla conoscenza, come per il bancomat; quindi gli smart token sono stati implementati con la stessa idea del bancomat, ovvero autenticazione su possesso e conoscenza.

**Domanda:** Ma non bastava utilizzare una smart card anche per il sito web?

**Risposta:** Sì, se tutti i computer avessero avuto un lettore.

Gli smart token fanno per i servizi web quello che il bancomat fisico fa per l'Atm, per prendere i soldi.

La password o sul token o sul sito sono equivalenti: alcune banche per l'accesso richiedono una password (che noi abbiamo) e il numero univoco del token. In altri sistemi il token aveva una tastiera alfanumerica che richiedeva una password; una volta inserita la password otteniamo la one-time password e questa veniva inserita sul sito. Questi sistemi sono equivalenti perché nel primo caso (autenticazione con la banca) la banca fa sia autenticazione basata sul possesso e conoscenza (perché chiede la password [conoscenza], e chiede la password del token; in questo caso il token ha solo un pulsante che lo deve premere il portatore). Quindi il token si può permettere di essere al portatore perché poi la password me la chiede il sito. L'altro tipo di token, quello con la tastiera, era più costoso. La password la si inseriva sul token (ovvero ci autentichiamo noi al token) ed esso restituiva la one-time password. Il procedimento è differente rispetto al primo esempio però l'utente mette sempre possesso e conoscenza.

## Funzionamento di uno smart token

Lo smart token funziona nel seguente modo:

- C'è un seme condiviso con il server;
- C'è dell'entropia esterna;
- Con seme ed entropia, viene calcolata, tramite un algoritmo misterioso, una **one-time password**.

Ovviamente lo scopo è che le password sono più lunghe, più robuste, meno mnemoniche; quindi più difficili da indovinare.

## Autenticazione basata su biometria

**Biometria = Possesso** di quelle caratteristiche biometriche, che sono univoche. Le caratteristiche biometriche possono essere:

- **Fisiche:** l'informazione biometrica tipica è l'impronta digitale, impronta della retina o del viso.
- **Comportamentali:** ad esempio possiamo considerare la *firma*, *timbro di voce*, *grafia*, *keystroke dynamics*, ovvero la dinamica della pressione dei tasti (cioè come premiamo i tasti, quanta pressione facciamo).

L'autenticazione biometrica è tecnicamente meno accurata, ma comunque più affidabile (più inattaccabile). Ma in realtà è tecnicamente meno accurata, a causa della rappresentazione digitale delle informazioni biometriche. La rappresentazione di una password è univoca (c'è un codice per rappresentare una password, il codice ascii); cioè se al posto di 'a' digitiamo 'e' la rappresentazione binaria cambia in virtù del codice che è univoco. Ma quando mettiamo la nostra impronta digitale, non mettiamo sempre la stessa impronta. Essa va campionata

per renderla digitale. Due campioni biometrici non saranno identici mai. Quindi la rappresentazione binaria della nostra impronta digitale non sarà “identica”, ma qualche bit (meno significativo) cambierà. Per poter capire se quell’impronta è proprio la nostra la dobbiamo inserire più volte (perché appunto non sarà mai la stessa; ma tutte queste volte che la inseriamo servirà a capire se siamo noi). Invece quando inseriamo una password questa basta inserirla una volta, perché la rappresentazione binaria è univoca.

Il funzionamento di autenticazione con biometria prevede i seguenti passi:

- Prima dobbiamo campionare, cioè dobbiamo registrare nel sistema un campione. Quindi si definisce un **template**, che è il template medio dei casi di campionamento.
- Per l’autenticazione, si prende l’impronta ‘live’ e la si confronta con il campione che era stato conservato. Anche qui l’aderenza non sarà al singolo bit. Quindi noi non metteremo mai un impronta che bit a bit è uguale, ma ci sarà un margine di tolleranza (ecco in che senso è la tecnica meno accurata)

Consideriamo la seguente figura:

Today's Biometric Signature from Cathy:	Cathy's Stored Biometric Pattern:	Tim's Stored Biometric Pattern:
389	390	284
416	418	570
501	502	534
468	471	501
353	355	399
Distance = 4 from that signature		Distance = 199 from that signature

A sinistra troviamo il campione live di dati di colui che si vuole autenticare, al centro abbiamo il template di dati, e a destra abbiamo il template registrato. Si vede che il campione live non è identico al campione registrato ma è molto simile (in questo caso l’accuratezza è minore). Con le password i bit devono corrispondere tutti.

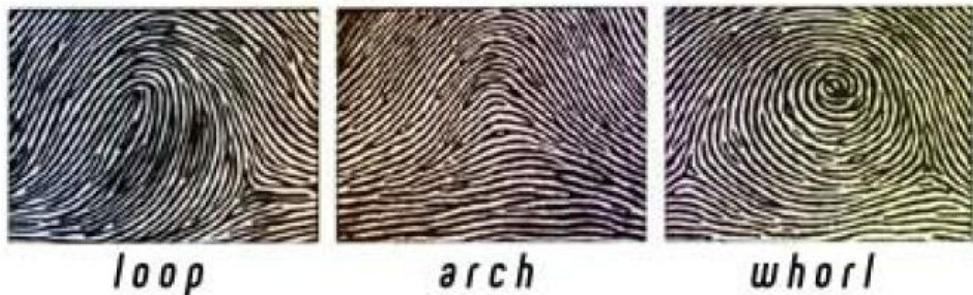
Si è molto discusso se questa è la tecnica più affidabile. In realtà è la tecnica meno protetta perché noi lasciamo impronte digitali dappertutto. Quindi se vogliamo la password di un utente dovremo “torturarla”; mentre con le impronte è più facile. Una volta si usava l’inchiostro per confrontare le impronte digitali:



Oggi chi fa **Vision**, si è posto il problema di come trasformare queste informazioni in binario.

Sono stati realizzati i tre schemi più diffusi di impronte digitali.

## *basic fingerprint patterns*



Fissati questi studi biologici, dal punto di vista informatico, si sono definite le **minuzie**. Esse rappresentano la fine o il punto di biforcazione di una linea (linee del nostro tessuto cutaneo). Riconoscere un'impronta si fa riconoscendo le coordinate delle minuzie.



# **CENNI DI CRITTOGRAFIA**

## NOTA

Assumiamo (**qui**) che la crittografia sia perfetta; quindi supponiamo che qualunque attaccante non riesca a violare il crittosistema. La inviolabilità del crittosistema la possiamo considerare equivalente al DY. Questa potrebbe essere un'ipotesi troppo restrittiva (che l'attaccante non possa rompere la crittografia); cioè una restrizione che semplifica il nostro discorso (e quindi lo rende meno realistico) o viceversa; una restrizione che potenzia il nostro discorso. Se diciamo che l'attaccante può far tutto ma la crittografia è perfetta, ci stiamo mettendo in un contesto semplificato, però c'è il discorso della separazione degli ambiti. Ad esempio: se abbiamo il cemento e dobbiamo costruire una casa e deve essere banalizzata la nostra capacità di costruire la casa, dobbiamo avere un cemento buono o un cemento depotenziato? Cioè, in questa similitudine, si vuole tesare la capacità di costruire le case, di progettare, di mettere i mattoni assieme. Se avessimo il cemento depotenziato e crolla la casa, possiamo dire che non è colpa nostra ma è colpa del cemento. Allora, nella sicurezza è la stessa cosa. Noi vogliamo progettare un sistema di sicurezza, un sistema sicuro, un protocollo. Quindi consideriamo il cemento migliore che c'è, così possiamo studiare il nostro protocollo di sicurezza (capire se è sicuro o meno) senza dover asserire un eventuale problema. In questo modo, con lo strumento migliore che c'è, possiamo misurare se riusciamo a progettare un sistema sicuro; perché altrimenti possiamo semplicemente dire che il nostro sistema sicuro non funziona perché è un problema di crittografia. Quindi è chiaro che considerare l'ipotesi, che la crittografia sia perfetta, è limitativa; ma se l'obiettivo è analizzare se il nostro sistema è sicuro o meno allora è un'ipotesi accettabile. Notare che, lasciata da parte la materia prima (la crittografia), poi possiamo fare tutte le ipotesi peggiori, cioè dato il cemento ottimo, il mattone ottimo, poi con pioggia, stravento ecc.. la casa deve reggere (quindi poi ci mettiamo nelle condizioni peggiori per capire se siamo stati in grado di costruire la casa). Quindi consideriamo la crittografia migliore che c'è per poter verificare che progetti di sicurezza funzionano.

## Fondamenti

La crittografia la possiamo considerare come la scienza di **criptare** (codificare) un testo in chiaro e **decriptare** (decodificare) il crittotesto associato:

- Un messaggio binario **m** può essere criptato in **c**, il quale a sua volta può essere decriptato in **m**.

Esiste una variante più moderna di crittografica, sviluppata nel 1977, detta **crittografia asimmetrica**. La crittografia tradizionale è detta **crittografia simmetrica**. C'è un vasto uso di crittografia in ambito militare e su Internet.

# Crittosistemi

Gli algoritmi di *encription* ( $\mathcal{E}$ ) e *decription* ( $\mathcal{D}$ ) prendono due parametri e restituiscono un particolare output. In particolare:

- L'encription prende in input un testo in chiaro e una chiave e dà in output un crittostesto:  $\mathcal{E}(\mathbf{m}, \mathbf{k}) = \mathbf{m}_k$ .
- L'operazione di decription prende in input un testo codificato e una chiave e da in output un testo in chiaro:  $\mathcal{D}(\mathbf{m}_k, \mathbf{k}) = \mathbf{m}$ .

Data una chiave  $K'$  e un crittostesto  $\mathbf{m}_k$ , questo viene decodificato come:  $\mathcal{D}(\mathbf{m}_k, K')$ , che produce  $\mathbf{m}$  se e solo se precise condizioni legano  $\mathbf{K}$  con  $K'$ . Cioè se abbiamo un crittostesto, l'unico modo per ottenere il testo in chiaro a partire dal crittostesto è quello di farne la decodifica per mezzo della stessa chiave. Se non sappiamo quale sia la chiave  $\mathbf{K}$ , allora proviamo con una certa chiave  $K'$ . Più  $K'$  si avvicina a  $\mathbf{K}$  e più il crittostesto decodificato sarà chiaro.

Quindi:

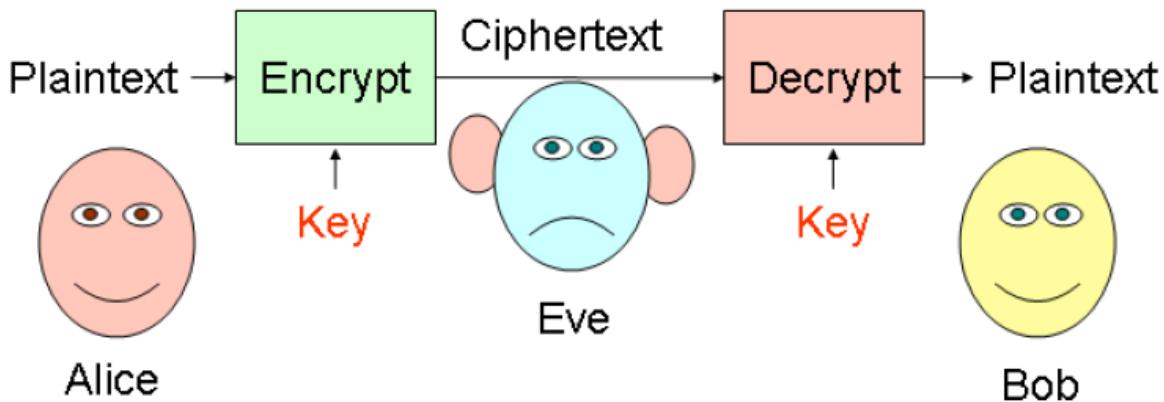
- Sia dato un testo in chiaro e una chiave  $\mathbf{K}$ . Possiamo applicare la funzione di encription,  $\mathcal{E}(\mathbf{m}, \mathbf{k}) = \mathbf{m}_k$ , e ottengo il crittostesto  $\mathbf{m}_k$ .
- Dato il crittostesto possiamo darlo in pasto all'algoritmo di decription.
  - Se la chiave è  $\mathbf{K}$  ottengo  $\mathbf{m}$ :  $\mathcal{D}(\mathbf{m}_k, \mathbf{k}) = \mathbf{m}$
  - Se non sappiamo quale sia  $\mathbf{K}$ , allora posso fare la decription con una certa chiave  $K'$  e ottengo un certo  $\mathbf{m}'$ :  $\mathcal{D}(\mathbf{m}_k, K') = \mathbf{m}'$ .

Le differenze che lega  $\mathbf{m}'$  e  $\mathbf{m}$  sono tanto evidenti quanto buono è il crittosistema. Quindi che differenza c'è fra  $\mathbf{m}'$  e  $\mathbf{m}$ ? Se l'algoritmo di encription è debole  $\mathbf{m}'$  si avvicina molto a  $\mathbf{m}$ . In alcuni casi, a seconda delle proprietà dell'algoritmo di encription,  $\mathbf{m}'$  potrebbe sovrapporsi a  $\mathbf{m}$  per certe porzioni; ovvero  $\mathbf{m}'$  potrebbe avere dei bit in comune con  $\mathbf{m}$ . Migliore è l'algoritmo di encription, più distante sarà il livello di  $\mathbf{m}$ .

Ma poiché inizialmente abbiamo supposto che, consideriamo che la crittografia sia perfetta, l'unico modo per ottenere  $\mathbf{m}$  da  $\mathbf{m}_k$ , è decodificare quest'ultimo con la chiave giusta  $\mathbf{K}$ .

Quindi l'associazione fra un testo in chiaro e un crittostesto dipende dallo specifico crittosistema e dalla chiave scelta.

Uno dei modi più semplici per fare encription è il **cifrario di cesare**. Perso un testo in chiaro, ad esempio "CIAO", presa una certa chiave  $K_1$ , e supponiamo di usare come algoritmo di encription il cifrario di cesare (questo algoritmo fa lo shifting). Se faccio un codifica per mezzo di questo algoritmo otteniamo "DLBP". L'attaccante, se riesce ad intercettare quest'ultima stringa, allora potrebbe iniziare, per riottenere il testo in chiaro, a fare lo shifting all'indietro (o in avanti); prima faccio uno shift di distanza 1, poi di distanza 2, di distanza 3, e così via. Cioè potrei fare una decription esaustiva, detta forza bruta, su tutte le possibili chiavi. Questo è un esempio di crittosistema banale, cioè poco robusto, violabile. Violabile vuol dire che dato un crittostesto, se non ho la chiave, uso una certa  $k'$ , riesco in qualche modo a ottenere informazioni sul messaggio; per esempio con una ciclicità di decription a tentativi.



Quindi se Alice deve mandare qualcosa a Bob e vuole evitare che un attaccante scopre questo messaggio, una soluzione è quella di fare un **encription** del messaggio con una certa chiave **Key**. L'attaccante vedrà il critttesto. Bob che ha la stessa **Key**, riuscirà a decriptare il messaggio ricevuto da Alice. L'ipotesi fondamentale che l'immagine evidenza è che il ricevente conosca la stessa chiave. Con questo procedimento c'è il problema di come passare la chiave; cioè come fa Bob a sapere che chiave ha applicato Alice? La risposta è descritta dopo 😊.

**Nota che:** Il crittosistema è noto a tutti, ovvero gli algoritmi di **encription** e **decription** sono noti a tutti. Ciò che viene tenuta nascosta, protetta, è la chiave di **encription** e no gli algoritmi.

Resta il problema di come condividere la chiave.

## Segretezza di una chiave

Se abbiamo un segreto c'è sempre quella probabilità minima che l'attaccante lo indovini ( $\Pr[\text{guess}(k)] = \frac{1}{2^{|K|}}$ , stiamo considerando in questo caso l'alfabeto binario e  $|K|$  rappresenta la lunghezza della chiave. Più lunga è la chiave, più difficile è indovinarla).

Questa definizione è stata utilizzata anche per il concetto di password.

**Osservazione:** Chiave e password sono la stessa cosa?

Sia la password che la chiave sono rappresentati da una stringa di bit, e hanno entrambi la peculiarità di dover essere tenute segrete, di andare in pasto ad algoritmi. Una password è una precisa istanza di una chiave, perché la password la posso usare come una chiave per fare l'encription. Quindi in quanto entrambi sono segrete e sono rappresentate da stringhe di bit, li possiamo considerare allo stesso modo. È possibile dire che la password solitamente la scegliamo da 8 caratteri, ( $\text{char: } 8 \times 8 = 64 \text{ bit}$ ), mentre una chiave è tipicamente maggiore di 64 bit (Ex 1024bit di chiave; per la password 1024 è un po' troppo esagerata). Quindi per una chiave si immagina qualcosa di più robusto, meno violabile. A parte queste considerazioni empiriche sulla lunghezza potremmo dire che entrambe sono stringhe di bit.

# Crittografia simmetrica

## Definizione

L'unico modo per estrarre il testo in chiaro da un crittostexto è decodificare quest'ultimo con **la stessa chiave** usata per costruirlo

La figura sopra la si può riassumere nel seguente modo:

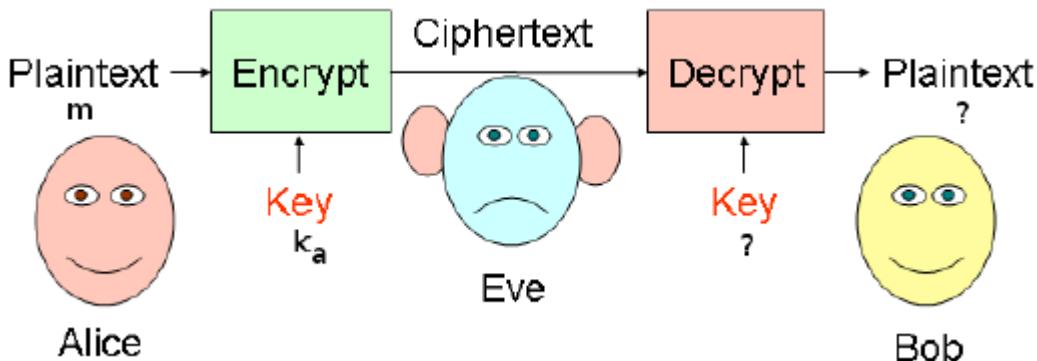
- $D(E(m, k), k) = m$   
La decription sul crittostexto con chiave  $k$  mi dà  $m$ .
- $\forall k_1: k_1 \neq k \Rightarrow D(E(m, k), k_1) \neq m$   
Se scelgo una chiave  $k_1 \neq k$  non otterrò il testo in chiaro.

## Crittografia simmetrica in rete

Fissato un agente A (macchina, utente), esso sia munito di chiave simmetrica, indicata come  $k_a$ .

$k_a$  è detta **chiave a lungo termine** perché il suo intervallo di validità è molto lungo, indipendente dalla specifica sessione di comunicazione.

Consideriamo di voler mandare un segreto a un interlocutore che sta in rete e di avere una chiave a lungo termine.



Alice ha la sua chiave a lungo termine (che potrebbe essere anche una nostra password  $k_a$ ), codifica un messaggio con  $k_a$  e lo manda a Bob. Bob ha il problema di sapere chi è  $k_a$  perché altrimenti non sarebbe in grado di decodificare il crittostexto ricevuto. Il problema che si ha è che, se anche trovassimo il modo di trasmettere la chiave in modo sicuro a Bob, non possiamo dare a Bob la chiave a lungo termine perché chiave/password mi impersonano; quindi la chiave a lungo termine per questo tipo di utilizzzi non si presta bene. Se ricevo qualcosa codificata con chiave  $k_b$ , in qualche modo devo avere questa chiave per decodificare il messaggio. Quindi mi servirebbe qualcosa, un sistema in cui tutti sanno le chiavi di tutti. Non va bene questo! **Una chiave a lungo termine è una risorsa sensibile**. Quindi, in sostanza, c'è un problema.

## Limiti della crittografia simmetrica

- **Limite 1:** A non vuole rivelare  $k_a$  a B, quindi B non può decriptare.
  - Sia  $k_{ab}$  una chiave dedicata specificatamente a questa sessione fra A e B.
  - $k_{ab}$  è detta **chiave a breve termine** o **chiave di sessione**.
  - Può essere condivisa fra i due agenti
- **Limite 2:** servirebbe una chiave di sessione per ogni coppia di agenti che vogliono comunicare fra loro
- **Limite 3:** come condividere  $k_{ab}$  fra A e B pur mantenendola confidenziale?

Manifestazioni del limite fondamentale della crittografia simmetrica: **come condividere il segreto iniziale su una rete insicura**

## Crittografia asimmetrica (1978)

### Definizione

- Ogni chiave  $k$  ha una sua **inversa** denotata  $k^{-1}$
- Ciascuna chiave **non** (problema computazionalmente intrattabile!) si può ricavare dall'altra, pertanto la coppia va generata monolicamente
- L'unico modo per estrarre il testo in chiaro da un crittotesto è decodificare quest'ultimo con **l'inversa della chiave** usata per costruirlo

Negli anni '70 fu inventata la crittografia asimmetrica. La crittografia asimmetrica supera il limite fondamentale della crittografia simmetrica.

Ogni sua chiave ha la sua inversa. Se Alice ha la chiave  $k$  allora la sua inversa è  $k^{-1}$ . Queste chiavi le possiamo generare insieme, ma data una chiave calcolare la sua metà, la sua inversa, sarà un problema computazionalmente intrattabile. L'unico modo per estrarre il testo in chiaro da un crittotesto fatto con la chiave  $k$  è fare la decodifica con l'inversa (possiamo anche scambiare  $k$  con  $k^{-1}$ ):

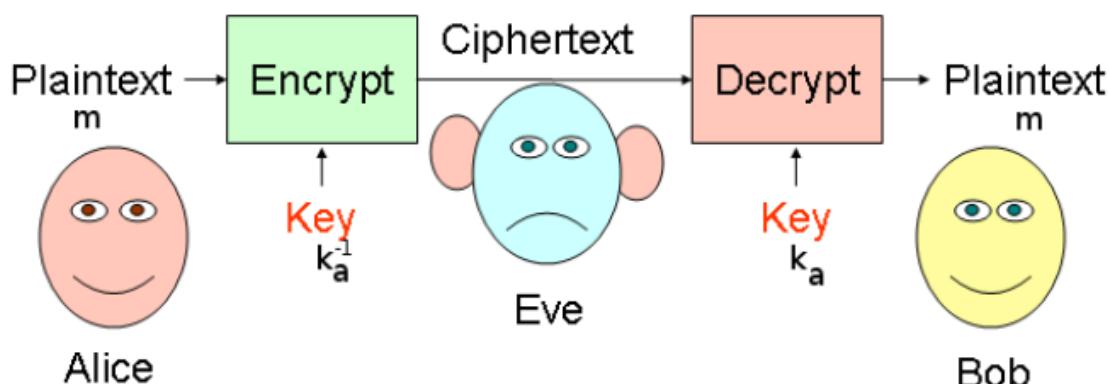
- $\mathcal{D}(\mathcal{E}(m, k), k^{-1}) = m$
- $\forall k_1: k_1 \neq k^{-1} \Rightarrow \mathcal{D}(\mathcal{E}(m, k), k_1) \neq m$

L'algoritmo di decription lo si può fare con qualsiasi chiave, ma per ottenere il testo in chiaro, visto che i crittosistemi si suppongono sempre perfetti, quindi crittoanalisi non ce nè, dato un crittotesto fatto con  $k$ , devo fare la decodifica con l'inversa di  $k$ .

Nel caso un cui  $k$  e la sua inversa coincidono, ci riconduciamo alla crittografia simmetrica. Tipica lunghezza di una chiave è 1024 bit.

Ci sono vari esempi di crittosistemi asimmetrici, come DSA, RSA.

## Crittografia asimmetrica in rete



La crittografia asimmetrica in rete risolve il problema descritto prima. Siccome ogni utente ha una coppia di chiavi, una se la tiene e l'altra la pubblica. Per convenzioni notazionali, la parte da pubblicare la indico senza il  $-1$  come apice della chiave e la parte da nascondere con il  $-1$  come apice della chiave. Se A pubblica  $k_a$  non mette a rischio  $k_a^{-1}$  (perché da una chiave non posso ricavare l'altra). Quindi supponiamo di aver pubblicato  $k_a$ . Se faccio  $\mathcal{E}(m, k_a)$ , per ottenere  $m$  dovrò fare  $\mathcal{D}(\mathcal{E}(m, k_a), k_a^{-1})$ . Ovviamente se avessimo reso pubblica  $k_a^{-1}$  e avessi fatto  $\mathcal{E}(m, k_a^{-1})$ , per ottenere  $m$  avremmo dovuto fare  $\mathcal{D}(\mathcal{E}(m, k_a^{-1}), k_a)$ .

Alice vuole mandare un messaggio segreto a Bob per mezzo di un sistema asimmetrico. Alice che chiave deve usare?



- Se Alice usa  $k_a^{-1}$  tutti possono decodificare
- Se Alice usa  $k_a$  nessuno può decodificare.

Alice usa la chiave pubblica del destinatario! In questo modo può decodificare solamente Bob (perché solo lui conosce l'altra relativa chiave):

- **Alice:**  $\mathcal{E}(m, k_b)$
- **Bob:**  $\mathcal{D}(\mathcal{E}(m, k_b), k_b^{-1}) = m$

Quindi il messaggio potrà attraversare la rete in maniera segreta tra il mittente e il destinatario. Con questo schema il problema della trasmissione segreta è risolta.

## Limite della crittografia asimmetrica

Il vero limite della crittografia asimmetrica è il seguente. Abbiamo Alice con chiavi  $(k_a, k_a^{-1})$ , Bob con chiavi  $(k_b, k_b^{-1})$ . Ma non abbiamo solo loro in rete. Potrebbe esserci un utente C con chiavi  $(k_c, k_c^{-1})$ , un utente D con un'altra coppia di chiavi, ecc.... Se Alice vuole comunicare in maniera segreta con Bob, Alice deve codificare con la chiave pubblica di Bob. Se Alice non applica la chiave di Bob ma la chiave di C, allora Bob, applicando la sua chiave privata, otterrebbe una stringa a caso. Non solo Bob non capirà niente, ma **c'è perdita di segretezza**, perché se C intercetta, applicando la chiave  $k_c$  riuscirebbe ad ottenere il testo in chiaro. Questo errore non bisogna farlo. Quindi ora Alice ha il problema di sapere qual è la chiave pubblica di Bob!

Per risolvere il problema della crittografia asimmetrica si usa una tecnica detta **certificazione**.

**Quindi:**

Restano due limiti da superare

- Crittografia simmetrica: come condividere il segreto iniziale fra una coppia di agenti che vogliono comunicare in maniera sicura
- Crittografia asimmetrica: come verificare il proprietario di una chiave pubblica (certificazione)

**Domanda:** Quando un crittosistema lo chiamiamo sicuro o perfetto?

Quando esiste solo una chiave per aprire il crittostesto; nel caso simmetrico la stessa con cui ho codificato, nel caso asimmetrico l'inversa di quella pubblica.

**Definizione** Crittosistema sicuro

- Sia calcolato  $\mathcal{E}(m, k)$  per ogni testo  $m$  e chiave  $k$ ;
- sia calcolato  $\mathcal{D}(\mathcal{E}(m, k), k_1) = n$  per ogni chiave  $k_1$  tale che  $k_1 \neq k$  se il crittosistema è simmetrico, o  $k_1 \neq k^{-1}$  se il crittosistema è asimmetrico;
- allora l'accesso a  $n$  **non aumenti significativamente** la probabilità di un attaccante di indovinare  $m$  o sue porzioni

# Hash crittografico

## Definizione:

- È facile calcolare l'hash, ma esso non è invertibile.
- Non è possibile cambiare il messaggio se prima non cambio l'hash. Cioè se ho  $m$  e  $m'$ ,  $\text{hash}(m)$  è diverso da  $\text{hash}(m')$  anche se  $m$  e  $m'$  si differenziano solo per un bit. Quindi a piccole modifiche di  $m$  corrispondono grandi modifiche nel risultato della funzione hash.
- Non riesco a trovare collisioni.

**Nota:** hash crittografico è diverso dall'hash studiato in algoritmi.

*Definizione accettabile da Wikipedia:*

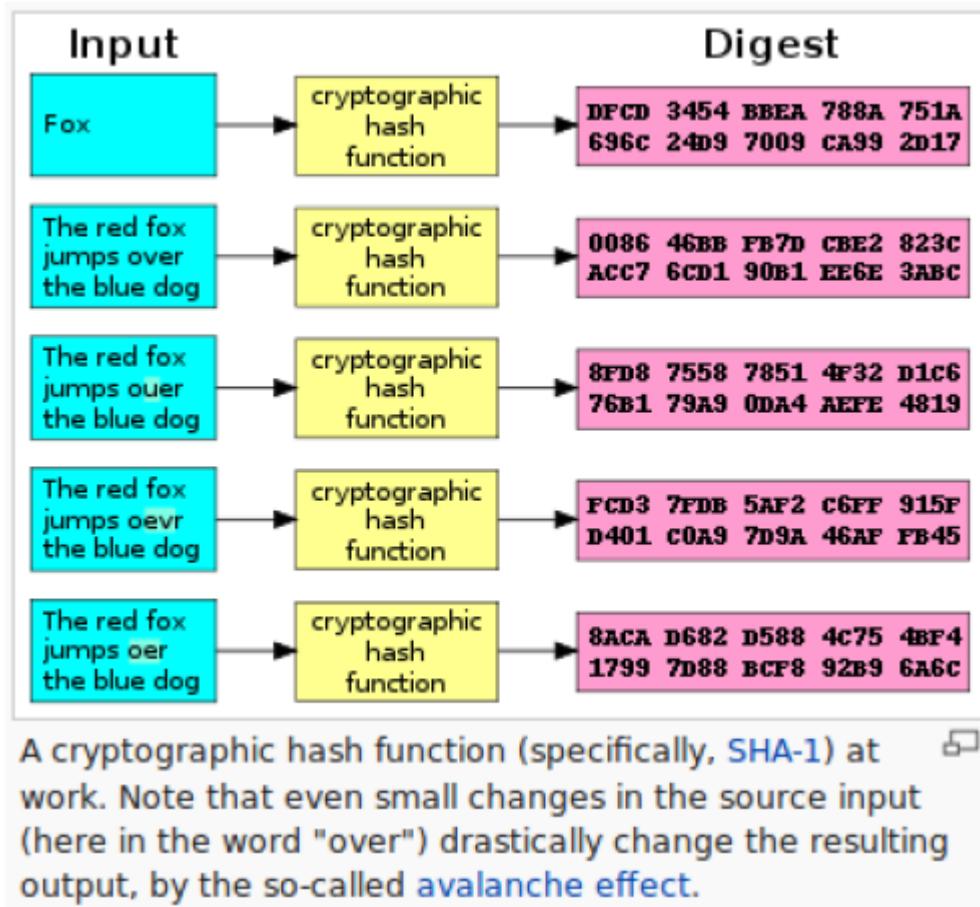
A **cryptographic hash function** is a **hash function** that takes an arbitrary block of **data** and returns a fixed-size **bit** string, the **cryptographic hash value**, such that any (accidental or intentional) change to the data will (with very high probability) change the hash value. The data to be encoded are often called the **message**, and the hash value is sometimes called the **message digest** or simply **digest**.

The ideal cryptographic hash function has four main properties:

- it is easy to compute the hash value for any given message
- it is **infeasible** to generate a message that has a given hash
- it is **infeasible** to modify a message without changing the hash
- it is **infeasible** to find two different messages with the same hash.

Cryptographic hash functions have many **information security** applications, notably in **digital signatures**, **message authentication codes** (MACs), and other forms of **authentication**. They can also be used as ordinary **hash functions**, to index data in **hash tables**, for **fingerprinting**, to detect duplicate data or uniquely identify files, and as **checksums** to detect accidental data corruption. Indeed, in information security contexts, cryptographic hash values are sometimes called **(digital) fingerprints**, **checksums**, or just **hash values**, even though all these terms stand for more general functions with rather different properties and purposes.

## Esempio: SHA-1

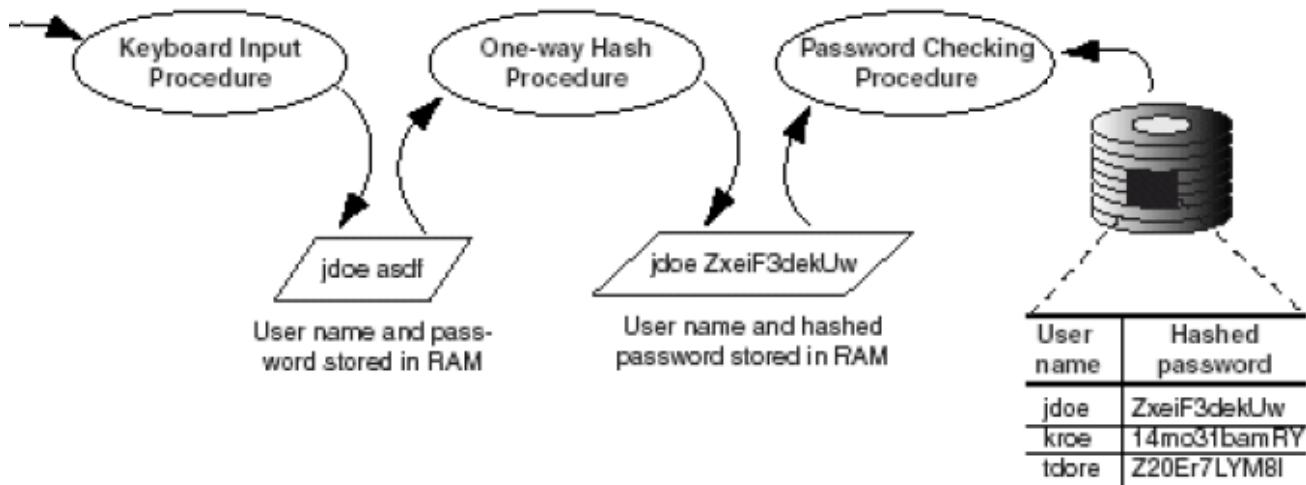


## Esempio di uso di hash: CTSS + hashing

Consideriamo il problema della rappresentazione in memoria della password. Una soluzione consisteva nell'avere una tabella dove ogni record rappresentava nome utente e relativa password. Un miglioramento di quest'ultima soluzione è: **CTSS + hashing** (1967). La password non è memorizzata in chiaro, ma è memorizzata in versione hash.

*Ma quindi l'hash funziona come l'encryption?* Se l'attaccante violasse il controllo di accesso vedrebbe una password in versione hash. Quindi la password non è codificata ma è in versione hash. **L'hash non è encryption.** L'hash è una funzione sostanzialmente lunaria, cioè se ho un messaggio faccio  $h(m)$ , non possibile successivamente "aprirla" per mezzo di una chiave; non la posso aprire ovvero invertire mai. Il crittore lo posso aprire se faccio decryption con la relativa chiave. **L'hash è un crittogramma buono e si presta a scopi di segretezza;** anziché dare il messaggio, do' l'hash del messaggio e nessuno lo può invertire e quindi si ottiene segretezza. Quello che si può fare è: supponiamo di avere un hash (non lo possiamo invertire (speriamo)), se abbiamo il messaggio in chiaro, potremmo farci sopra un hash e verificare se corrisponde al primo. Quest'ultimo problema è preciso al problema del controllo della password. La password sta in memoria protetta da hash. L'utente arriva, mette la password e il sistema calcola l'hash e vede se corrisponde con quello in memoria. Se l'utente inserisce un solo bit diverso

l'hash va per i fatti suoi. Quindi se il confronto mi dà Ok, vuol dire che l'utente ha inserito la password giusta.



## Salting

**Salt** indica quel messaggio random aggiunto a una password per proteggerla da attacchi dizionario prima che vi si applichi l'hash.

Il salt era storicamente di 12 bit. Essi facevano la differenza. Quindi una password banale potrebbe essere arricchita con 12 bit random. L'utente non deve ricordarsi la sequenza random aggiuntiva di 12 bit poiché il salt se lo gestisce il sistema. Il salt serviva a irrobustire la password per attacchi dizionari. Fino a un paio di anni fa questo tipo di irrobustimento era buono (quindi l'aggiunta di questi 12 bit). Oggi non va più bene.

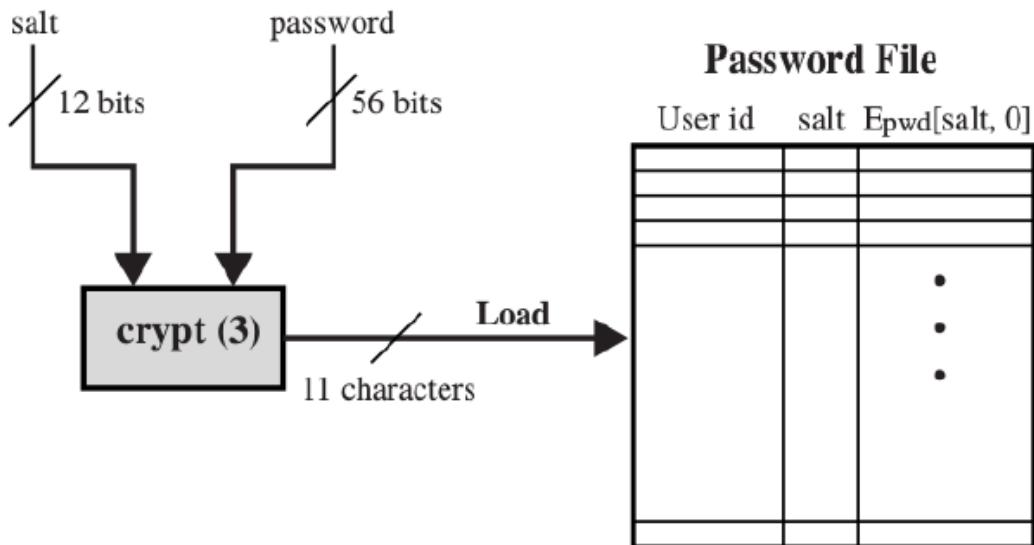
In Unix, funziona così:

- Genera salt per ciascuna nuova password da memorizzare. Appena facciamo add User e ci chiede la password, il sistema ha già generato il salt random. Questi bit vengono aggiunti alla password.
- Usa la password per codificare una stringa di zeri insieme a salt.

Quindi in UNIX non si usano funzioni hash.

Abbiamo sia la password che il salt. L'idea di Unix è che genera il salt e usa la password come chiave di encryption. Come segreto uso la password. Quindi la password viene usata come chiave di encryption. Encryption di una stringa di zeri, la cui lunghezza è data dalla funzione **crypt(3)** che è una incarnazione di **DES**. Questa funzione di encryption, prende come primo parametro la stringa di bit + salt (di cui facciamo la concatenazione), e come secondo parametro la password. Quindi viene codificata quella stringa concatenata per mezzo della password (per proteggere un segreto poiché non abbiamo un altro segreto allora usiamo la password come chiave di encryption).

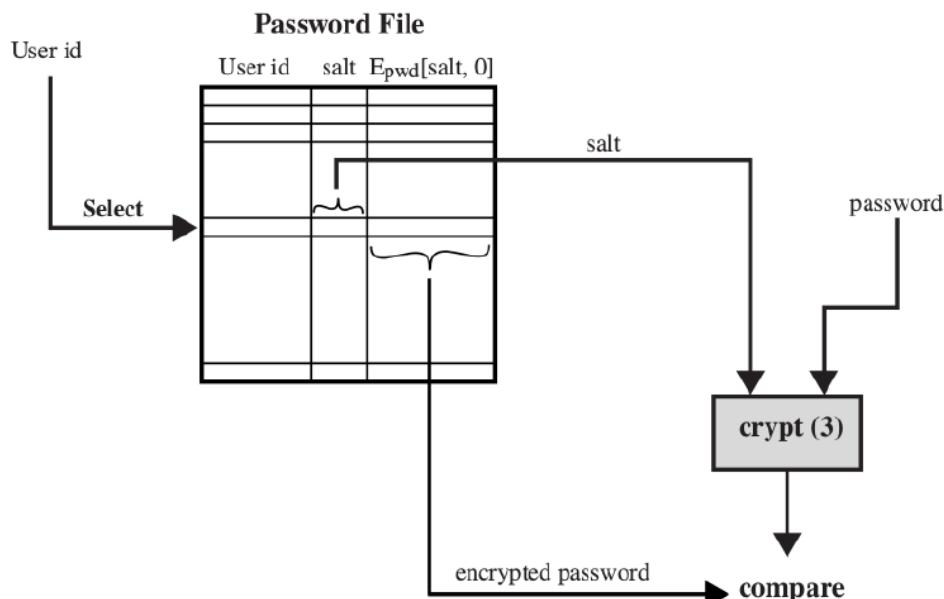
## Unix: aggiunta di una password



L'utente inserisce la password. Essa insieme al salt, viene data in pasto alla funzione crypt(3), la quale produce un certo numero di caratteri. Essi vengono memorizzati in quel file di password.

Questo sistema è diverso dal CTSS + hashing, perché memorizza lo user id, insieme al salt e insieme all'encryption per mezzo della password della stringa salt più certi zeri. Quindi c'è l'utilizzo di una funzione di encryption a modi hashing. Quando l'utente va ad inserire la password, il sistema può decodificare l'encryption. Il sistema riesce a capire che la password è quella giusta se riottiene la stringa di bit più il salt. La randomicità del salt diventa importante.

## Unix: verifica di una password



Quando l'utente si vuole autenticare, inserisce User id e la password. Il sistema cerca lo User Id e trova il salt e l'encription. Il salt insieme alla password dell'utente

viene data a crypt(3) e si fa il matching. Crypt(3) è una funzione di encryption, ma viene usata per nascondere il segreto nell'encryption in maniera molto originale e questo matching è l'algoritmo tipico che noi usiamo per le funzioni hash.

## Freshness

La **Freshness**, più che una proprietà di sicurezza è una meta-proprietà di sicurezza, ovvero ottengo un attributo delle proprietà di sicurezza, qualcosa che può valere su altre proprietà.

In un tempo infinito un attaccante potrebbe violare qualsiasi cosa, perché tutte le proprietà di sicurezza che facciamo sono tutte computazionalmente limitate. Se c'è una chiave che usiamo da molto tempo e una invece appena creata, quella più affidabile è quella che abbiamo appena generato.

Consideriamo l'autenticazione. Faccio una sessione e autentico il mio interlocutore X. Se il mio interlocutore è X non è detto che l'istante successivo lo sia sempre lui. In particolare non è detto che se io adesso autentico X, X sia stato autenticato con me 10 minuti fa. Cioè dall'altra parte ci deve essere l'interlocutore al momento giusto. Ad esempio, se voglio comprare per 1 euro un foglio di carta dall'utente A, l'utente A lo sto autenticando; ma se dopo un attimo, nel momento in cui sto dando l'euro c'è un attaccante allora in questo caso mi hanno fregato.

Quindi quando noi facciamo una qualunque sessione di comunicazione, interazione, di fatto stiamo autenticando il nostro interlocutore continuamente.

Immaginiamo il bancomat. Esso autentica un riccone. Dopo 10 minuti se ci vado io, il bancomat, mi deve far rifare l'autenticazione, perché altrimenti posso prelevare i soldi del riccone.

La freshness è una proprietà di sicurezza è una meta-proprietà sensibile? Cioè che ha un valore significativo in termini di sicurezza e pertanto potrebbe attrarre l'interesse malevolo di un attaccante? È una risorsa sensibile, perché consideriamo un sistema di autenticazione che non abbia il sistema di freshness. Esso fallisce. Potrei aspettare che il riccone se ne va, non autenticarmi, e sfruttare la sua di autenticazione (se essa non ha un intervallo di validità limitata). In particolare c'è un altro tipo di attacco. Il fatto che un attaccante riesca a spacciare delle credenziali di autenticazione obsolete, antiche, come recenti. Cioè il riccone si autentica (10 minuti fa); l'attaccante intercetta (in modo passivo del termine, stare ad origliare, ascoltare) le credenziali, registra il traffico. Se poi dopo un paio di minuti l'attaccante replica il traffico (non necessariamente deve essere la password) si potrebbe spacciare per il riccone.

Quindi se non c'è un meccanismo di freshness, l'attacco di replica sarà sempre possibile.

La freshness si può fare in due modi: **timestamp** e **nonce**.

Il timestamp è un marcitore temporale. Importante per il timestamp è la sincronizzazione.

Il **nonce** è stato inventato nel '77 da due tizi. La loro idea era quella di creare un numero random e di usarlo solo una volta. ("numero random that is used only once").

Una **one time password (OTP)** è un numero usato solo una volta (pseudorandom). Si può usare come freshness perché cambia continuamente. C'è una sincronizzazione preconcordata e quindi ciascuno dei due interlocutori sa questo OTP a quale intervallo si riferisce.

# **PROTOCOLLI DI SICUREZZA BASILARI**

## Protocolli basilari per la freshness

I protocolli basilari per la freshness sono:

- **Timestamp**

- Chi vuol dare la garanzia di freshness inserisce il timestamp (pertanto potrebbe barare), associandolo al messaggio target in maniera affidabile (altrimenti chiunque altri potrebbe attaccare)
- Gli orologi distribuiti devono essere sincronizzati.

Quindi non solo l'associazione deve essere affidabile, ad esempio, inventiamo una chiave di sessione e vogliamo applicargli il timestamp. Applichiamo il timestamp perché all'interlocutore non gli serve conoscere il solo timestamp. Il problema dell'associazione c'è sempre. Io mando un timestamp protetto. All'interlocutore gli arriva questo timestamp e questo vuol dire che è stato mandato il messaggio a quel tempo lì. Per scopi di autenticazione c'era l'interlocutore a quel tempo. Ma vogliamo applicare la freshness alla chiave (ovvero vedere che la chiave è "fresca"), affianco alla chiave mettiamo il timestamp. In questo modo nasce il classico problema di associazione sicura perché l'attaccante potrebbe cambiare il timestamp. Allora, per risolvere il problema dell'associazione sicura, è possibile usare uno strumento crittografico.

- **Nonce**

- Chi vuole ricevere la garanzia di freshness manda la nonce (questo potrebbe essere un numero random) e aspetta traffico che citi la nonce, e questo traffico risulterà posteriore all'istante di creazione della nonce.

### ESEMPIO Timestamp & nonce

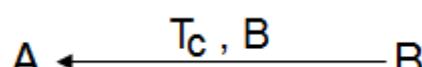
A autentica B.



Se A autentica B significa che B ha dimostrato di esserci di recente. Ovvero B precedentemente ha usato, ad esempio, un timestamp. Sia  $T_c$  il tempo corrente.

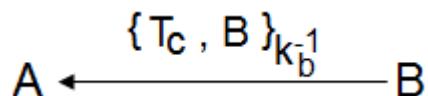


Questo non è un protocollo di autenticazione di B con A con freshness, ma è un timestamp! E quel timestamp lo può mandare chiunque. Quindi bisogna dire ad A che veramente lo sta mandando B. Questo si può fare, per esempio, concatenando  $T_c$  con l'identità.

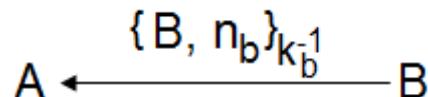


Ma anche in questo modo non stiamo ottenendo un protocollo di autenticazione di B con A con freshness, nonostante il messaggio dica ad A che c'è B a tempo  $T_c$ . Questo perché l'attaccante DY potrebbe alterare il messaggio, al posto di B potrebbe scrivere C, oppure potrebbe lasciare B e mettere  $T'_c \neq T_c$ , e quindi questo protocollo nei confronti di un attaccante DY è attaccabile.

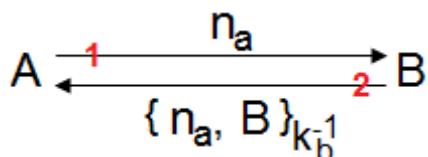
L'associazione con l'informazione sensibile di questo protocollo (quest'ultima essendo l'identità di B) è convalidata da uno strumento crittografico. Solo B può costruire, ergo autenticazione; chiunque può decodificarlo, in particolare A. Quindi con i timestamp, la storia è facile:



Con le nonce non ci sono orologi condivisi. B può inventare numeri random  $n_b$ . B potrebbe usare  $n_b$  come identificativo, quindi potrebbe fare, usando uno strumento crittografico:  $\{B, n_b\}$ .



Non è la stessa cosa di prima! La soluzione di prima riguarda un marcatore temporale con orologi sincronizzati; questo è un numero random. A riceve questo numero random. Ma non è che questo numero random può dire ad A che dall'altra parte c'era B e c'era di recente. B quando inventa  $n_b$  si può annotare l'istante in cui lo ha inventato (timestamp). Però non risolviamo il problema, perché chi ha inventato la nonce non è colui che deve dare garanzie di freshness, ma è chi se la deve ricevere. Supponiamo che A inventi  $n_a$  e si annota l'istante in cui lo ha inventata e la manda a B. B si deve autenticare e manda indietro B ed  $n_a$  con encryption (perché se non si mette encryption l'attaccante potrebbe cambiare B con C):  $\{B, n_a\}_{k_b^{-1}}$ .



L'encryption non intende proteggere la nonce, ma intende associare univocamente B come identità alla nonce. A decodifica questo messaggio (con la chiave pubblica di B) e vede che B si vuole autenticare. A capisce che B si vuole autenticare con lui perché riconosce  $n_a$ . Allora possiamo dire che: ciò che A riceve, ovvero il messaggio 2 che B manda per autenticarsi, risulterà posteriore all'istante in cui A ha creato la nonce. Questo è vero perché la nonce è random. Cioè se è random il suo utilizzo può essere anteriore rispetto a quando è stata creata? Prima non la conosceva nessuno!

**Domanda:** Perché la nonce non deve essere protetta?

Perché è random e la sua segretezza ha un valore puramente temporale. Ma non è una segretezza che dopo l'istante della creazione serve più. Dopo che è creata, la segretezza non serve più; tanto è vero che dopo la creazione A la manda in chiaro.

## Dalla crittografia alla sicurezza

La crittografia è uno strumento di sicurezza (non è la sicurezza). Per passare dalla crittografia alla sicurezza, ovvero a risolvere i problemi reali, si costruiranno delle cose che utilizzano la crittografia e detti **protocollo di sicurezza**. Più avanti verranno descritti vari esempi di protocollo di sicurezza, ovvero algoritmi distribuiti che prevede lo scambio di vari messaggi critografici fra agenti remoti, beneficiari della sicurezza che esso garantisce.

Assunzioni **massimamente stressanti** per il protocollo: crittosistema sicuro e (almeno) modello di attaccante DY. Questo vuol dire, prendere il peggior modello di attaccante, di rischio, per un edificio (per esempio), allor quando però ci sia a disposizione cemento che funziona bene.

## Sintassi dei messaggi critografici

Nei protocolli di sicurezza tratteremo:

- **Nomi di agenti:** salve sono Pulvirenti. Quindi è un nome di un agente (A, B, C, ... ) che si vuole autenticare, che vuole una garanzia di autenticazione, una qualche garanzia di sicurezza.
- **Chiavi crittografiche:** viste già nel capitolo “cenni di crittografia”. Si è visto che usiamo come pedice della chiave un nome per indicare il proprietario della chiave. Quindi  $k_a$  è la chiave di **A**. Con un solo pedice, tipicamente indicheremo una chiave a lungo termine. Abbiamo anche introdotto il concetto di chiave inversa.
- **Nonce:** abbiamo indicati con  $N_a, N_b, \dots$  e ancora una volta il pedice mi indica il proprietario.
- **Timestamp:** è un indicatore temporale. Sono stati indicati con  $T_a, T_b, \dots$  il pedice mi indica l'autore.
- **Digest:** è il risultato dell'applicazione di una funzione hash:  $h(m), h(n) \dots$
- **Etichette:** “Trasferisci \$100 dal conto di...”. Ex: Sia **E** un etichetta saliente del tipo “trasferisci i soldi da **A** a **B**” e sia  $m_k$  un altro messaggio saliente (potrebbe essere una nonce o altro). Con la virgola indichiamo la **concatenazione**. Consideriamo il seguente messaggio **E**,  $m_k$ . Ma in particolare consideriamo l'encription. Il messaggio in questo modo è costruito prima per encription e poi per concatenazione. Un messaggio di questo genere, dal punto di vista della sicurezza, che valore/problemi ha? Supponiamo che la segretezza non sia un requisito richiesto; quindi voglio dire a tutti che sto trasferendo soldi a Bob. Ma in questo modo l'etichetta la potrebbero cambiare tutti. Se invece facciamo questa operazione  $\{k, m\}_k$

(l'encrption la applico alla fine), ne deriva che l'etichetta e il messaggio sono vincolate insieme. È chiaro che l'attaccante potrà sempre fare una violazione di integrità; quindi prende il crittotesto, non ci capisce niente, e fa una violazione di integrità. Ma non sarà una violazione di integrità utile, perché una violazione di integrità utile è del tipo: prendo il messaggio originale "trasferisci 10 euro da me a B",... e quindi una violazione di integrità utile è "trasferisci 110 euro da me a B",... che realizza un attacco. Se invece l'etichetta sta dentro il crittotesto, abbiamo una stringa di bit, l'attaccante vedrà una stringa non comprensibile, ma potrà sempre flippare una bit da 0 a 1; ma in questo modo altererà il crittotesto in una maniera tale che il ricevente non se ne potrà fare nulla. Questo ha più le sembianze di un attacco distruttivo di negazione del servizio; non di un attacco come nel primo caso di appropriazione indebita di fondi.

Notare che la concatenazione non è un'operazione crittografica. Quindi se abbiamo un messaggio del tipo

$n, m, p$

dobbiamo sempre assumere che questo messaggio possa essere alterato dall'attaccante per esempio così

$n, m, p'$

perché la concatenazione lascia le componenti separabili e alterabili. E se una di queste componenti fosse un crittotesto? Se  $p$  fosse un crittotesto potrebbe essere alterato in  $p'$  e quindi ricorsivamente vale quello detto prima, ovvero se è un crittotesto esso potrebbe essere alterato in maniera brutale da non poter essere conosciuto nemmeno dal ricevente; se invece è in chiaro "trasferisci 10 euro", questo potrebbe essere alterato in maniera saliente come "trasferisci 100 euro". Questo discorso lo si può ripetere su ciascuna delle componenti, perché quelle dette prima non sono operazioni crittografiche, quindi chiunque può alterarle.

**Domanda:** Si può autenticare un messaggio concatenato? Cioè se ricevo un messaggio  $m, p$  ha senso chiedersi chi sia il mittente di questo messaggio?

**Risposta:** Poiché la concatenazione la può alterare chiunque, allora se mi arriva  $m, p$  mi chiedo ma è veramente Pulvirenti? In quanto c'è concatenazione non la potrò mai autenticare. Questo perché non è una operazione crittografica, quindi  $m, p$  potrebbe essere alterata in  $m, q$ .

## Sintassi di un protocollo di sicurezza

La sintassi di un protocollo di sicurezza è del tipo:

- *Numero passo, mittente, freccia (spedizione), ricevente, due punti, messaggio crittografico (da ripetere per ciascun passo del protocollo):*

1.  $A \rightarrow B : A, N_a$

2.  $B \rightarrow A : N_{ak_b^{-1}}$

Dove c'è la freccia specifichiamo sia mittente che ricevente. L'etichetta mittente non è mai significativa, nella misura in cui è alterabile. Anche il destinatario è

alterabile, però esso indica chi deve fruire del messaggio; nel senso che studieremo i messaggi dal punto di vista del destinatario. Se B riceve il messaggio che può farci! Se un agente spedisce un messaggio ad un altro non ha garanzie che quest'ultimo lo riceva. L'associazione del messaggio non garantisce che il messaggio sia recapitato.

C'è però una parte nascosta nella notazione. Quando il messaggio1 viene ricevuto da B, che ci fa B? Nella notazione non è specificato. Ad esempio B potrebbe fare questo: se riceve il messaggio1 scomponere la prima e la seconda componente, prende la seconda componente e applica l'encrption con la sua chiave privata. Il passo 2 indica solo questa spedizione; non indica i passi che ci stanno dietro per costruire il messaggio 2.

## Protocolli basilari per la segretezza

Come mandare il messaggio segretamente da A a B?

Consideriamo il caso **simmetrico**:

- A manda a B un messaggio codificato con una chiave condivisa. Prerequisito è che i due agenti conoscano la chiave  $k_{ab}$ .

Questo protocollo è sicuro? Questa è la domanda che dobbiamo porci davanti ogni protocollo. Sicuro vuol dire, raggiunge il suo obiettivo dichiarato? In questo caso è la segretezza del messaggio **m** nei confronti di B. Non può essere che l'attaccante DY possa scoprire la chiave  $k_{ab}$ ? Nei prerequisiti questo è escluso, perché solo A e B conoscono la chiave di sessione.

Consideriamo il caso **asimmetrico**:

- Nel caso asimmetrico si può ottenere un protocollo di segretezza per un certo messaggio **m**. I prerequisiti sono i seguenti:
  - B abbia una chiave privata valida (sicura, non scaduta)
  - A possa verificare che  $k_b$  è di B.

A conosce almeno 3 chiavi: la sua privata, la sua pubblica e quella pubblica di B.

Questi protocolli basilari **non hanno schemi di freshness incorporati**. Quindi per questo motivo questi protocolli saranno sempre soggetti a un attacco; sono vulnerabili alla replica perché non c'è una strategia di freshness.

## Protocolli basilari per l'autenticazione

Consideriamo il caso **simmetrico**:

- Somiglia molto a quello descritto prima; solo che nel caso precedente c'era **m**, mentre qui abbiamo un messaggio più specifico. Dentro il crittotesto possiamo mettere qualsiasi cosa, perché l'autenticazione di A con B è realizzata dall'operazione crittografica per mezzo della chiave condivisa.

Quindi nel modo più banale, io specifico chi sono: **A**. Dichiaro la mia identità. Solo che la dichiarazione della propria identità, dal punto di vista della sicurezza, non è affidabile. Chiunque potrebbe dichiarare una falsa identità. Quindi non è il contenuto del messaggio, ma è l'encryption con quella precisa chiave che realizza l'autenticazione. Il destinatario decodificherà, e autenticherà **A** sulla base del fatto che sa, per ipotesi, che la chiave di sessione  $k_{ab}$  è condivisa con **A**. Da qui possiamo notare, ancora una volta, il problema di associazione, perché se il destinatario creasse una chiave di sessione da utilizzare con **C**, rischierebbe di autenticare **C**, ma questo problema è eliminato dall'ipotesi.

Riassumendo, nel caso simmetrico i prerequisiti sono i seguenti:

- **Prerequisito1**: Chiave  $k_{ab}$  sia condivisa fra A e B soli
- **Prerequisito2**: B possa verificare il Prerequisito1

$$A \rightarrow B : "Sono io!"_{k_{ab}}$$

Consideriamo il caso **asimmetrico**:

- I prerequisiti sono i seguenti:
  - **Prerequisito1**: A abbia una chiave privata valida
  - **Prerequisito2**: B possa verificare che  $k_a$  è di A, il che vuol dire che c'è certificazione.

$$A \rightarrow B : "Sono io!"_{k_a^{-1}}$$

## Combinare segretezza e autenticazione

È possibile combinare i prerequisiti descritti prima.

Consideriamo il caso **simmetrico**:

- **Prerequisito1**: chiave  $k_{ab}$  sia condivisa solo fra **A** e **B**
- **Prerequisito2**: B possa verificare il Prerequisito1

$$A \rightarrow B : m_{k_{ab}}$$

Se B riceve quel messaggio, B potrà decodificarlo, e se consideriamo il prerequisito 1 solo B può decodificarlo, quindi questo messaggio arriverà a B in maniera segreta. Con il prerequisito 2, B sa con chi condivide la chiave. Quindi questo messaggio, al tempo stesso, autenticherà il mittente.

Consideriamo il caso **asimmetrico**: (\*)

In questo caso abbiamo 4 prerequisiti, nonché sono la somma di quelli che abbiamo visto prima:

- **Prerequisito1**: B abbia una chiave privata valida
- **Prerequisito2**: A possa verificare che  $k_b$  è di B
- **Prerequisito3**: A abbia una chiave privata valida
- **Prerequisito4**: B possa verificare che  $k_a$  è di A

Con questo schema, per ottenere la segretezza dobbiamo codificare la chiave pubblica del destinatario e per ottenere l'autenticazione usiamo la chiave privata

del mittente, quindi per ottenere il caso (\*) basta iterare le due operazioni. Le 2 operazioni descritte prima li possiamo fare in maniera intercambiabile; cioè prima usiamo la chiave pubblica del destinatario e poi quella privata del mittente o viceversa:

$$1. A \rightarrow B : \{m_{k_a^{-1}}\}_{k_b} \quad o \quad 1. A \rightarrow B : \{m_{k_b}\}_{k_a^{-1}}$$

Dal punto di vista di notazioni crittografiche fa differenza quest'ultima notazione.

Nei protocolli appena descritti **non ci sono schemi di freshness**.

## Come ottenere integrità?

I protocolli descritti prima sono alterabili brutalmente, a causa di attacchi di integrità. In generale si può fare, a meno che non ci sia uno strumento realizzato in modo da impedire attacchi di integrità. Un approccio potrebbe essere l'utilizzo di un qualche checksum del tipo usato nei protocolli di rete. Qualunque checksum sarà violabile da un attacco che viola i contenuti e il checksum, perché il checksum si può sempre ricostruire, ricalcolare!

Per risolvere questo problema possiamo usare le funzioni hash per il checksum. Queste funzioni comprimono l'input in un output di dimensione ben fissata.

$$1. A \rightarrow B : m, h(m)$$

L'attaccante potrebbe alterare  $m$  in  $m'$  e poi calcolare l'hash di  $m'$  e il ricevente si vede  $m'$  con l'hash di  $m'$ . Ma anche in questo caso c'è stato un attacco di integrità. Per risolvere questo problema, possiamo usare un hash crittografico, ovvero un hash arricchito con un encryption. Criptiamo l'hash con la chiave privata del mittente:

$$1. A \rightarrow B : m, h(m)_{k_a^{-1}}$$

Questo procedimento può funzionare? Supponiamo che l'attaccante intercetti il messaggio. Supponiamo che il messaggio sia mandato in questo modo:  **$m$ , checksum**. Come detto prima questo schema fallisce, perché l'attaccante potrebbe alterare  **$m$**  in  **$m'$**  e poi calcolare **l'hash di  $m'$**  e il destinatario non noterebbe nulla. Se consideriamo l'hash crittografico, il destinatario dovrà estrarre la prima componente  **$m$**  e si dirà "si direbbe che qualcuno voglia mandarmi questo messaggio  $m$ ". Il destinatario vuole vedere chi è questo qualcuno e vedere se il messaggio è stato alterato. Allora, il destinatario, prende la seconda componente e gli applica la decryption per mezzo della chiave pubblica di A, tirando fuori l'hash di  $m$ . Il destinatario calcola l'hash della prima componente e confronta quest'ultimo risultato con la seconda componente estratta. Se sono uguali allora il destinatario capisce che il messaggio è integro, non è stato alterato, altrimenti il messaggio viene rifiutato perché è stato alterato.

Ma il principio di questo algoritmo è quello del checksum. Riceviamo il messaggio e il suo checksum, così possiamo ricalcolare il checksum del messaggio e vedere se è uguale a quello di prima. Per questo motivo viene ricalcolato l'hash. Se il checksum originario potesse essere alterato dall'attaccante questo schema di integrità fallirebbe. Fallirebbe nel senso che il destinatario potrebbe essere imbrogliato nell'accettare un certo  $m'$ , quando invece era stato spedito un certo  $m$ . Se invece il checksum non è alterabile dall'attaccante, allora questo schema funzionerebbe come descritto prima, perché il checksum viene protetto crittograficamente e quindi è il termine di confronto perfetto (perfetto nel senso che è inalterato, intoccabile) per il messaggio. Quindi se abbiamo il checksum inalterato, ogni volta prendo il messaggio e lo confronto. Poiché il checksum è in versione politica dell'hash, ovvero prima faccio la decodifica e in questo modo ottengo la versione hash. Ma il messaggio non ce l'ho in versione hash; applico a quest'ultimo l'hash e vedo se è uguale a quello ottenuto precedentemente (anche se è stato modificato un singolo bit dal messaggio originale, il confronto dei due hash avrà esito negativo!).

Questo schema funziona non tanto perché è uno schema di integrità puro, ma perché ottiene allo stesso tempo integrità e autenticazione (tanto è vero che si rifà alla chiave privata del mittente).

Questo schema si chiama **firma digitale**. La firma digitale si fa in questo modo: prendiamo in messaggio e gli scriviamo accanto la versione codificata di una nostra chiave.

$$\text{sign}_A(m) = m, h(m)_{k_a^{-1}}$$

Questa trupla è la **firma digitale**.

## Firma digitale

La firma digitale è basata su crittografia asimmetrica, quindi ha bisogno di certificazione delle chiavi pubbliche.

Ma se la firma digitale è quella descritta prima, che differenza c'è tra firmare e fare la codifica? L'algoritmo di firma utilizza l'encrption, quindi che differenza c'è tra verificare la firma e decodificare? L'algoritmo descritto precedentemente è l'algoritmo di verifica di una firma ed è più ampio di fare una decodifica.

Quindi la firma digitale consiste in due algoritmi, uno di **creazione** e uno di **verifica**. **Creazione è più complesso di codificare e verificare è più complesso di decodificare.**

Ma allora, la firma digitale, possiamo dire che è l'omologo digitale della firma a penna?

Con la firma digitale si ottiene sia integrità del documento che autenticazione del firmatario, contrariamente alla firma cartacea. La firma cartacea è facilmente modificabile/falsificabile (chi non ha mai falsificato una firma a scuola!). Per esempio se scriviamo il numero 19, allora esso è alterabile facilmente in 18; il numero 10 è facilmente alterabile in 100.

Con la firma digitale si ottiene qualcosa di più della firma a penna. Con la firma a penna otteniamo solo l'autenticazione. Ma in realtà se falsifichiamo la firma cartacea, non otteniamo una alterazione di integrità, ma una alterazione di autenticazione. Mentre se ad esempio, in una giustificazione a scuola, se veniva messa come data 18, e specificata la firma del genitore, se alteriamo 18 a 20, questa è una alterazione di integrità.

Quindi, la firma a penna non garantisce l'integrità, mentre la firma digitale si, perché se noi cambiamo un bit nel messaggio, chi lo riceve si accorgerà di questa modifica.

La firma a penna viene utilizzata in **WATA**. In questo protocollo viene usata una firma a penna sul token a sugello di un'associazione di sicurezza. Ma in questo protocollo funziona bene la firma a penna, perché è messa proprio idealmente su tutta l'informazione. Quindi se la firma a penna potesse ricoprire l'intero certificato di carriera universitaria, allora il certificato potrebbe essere integro, altrimenti no.

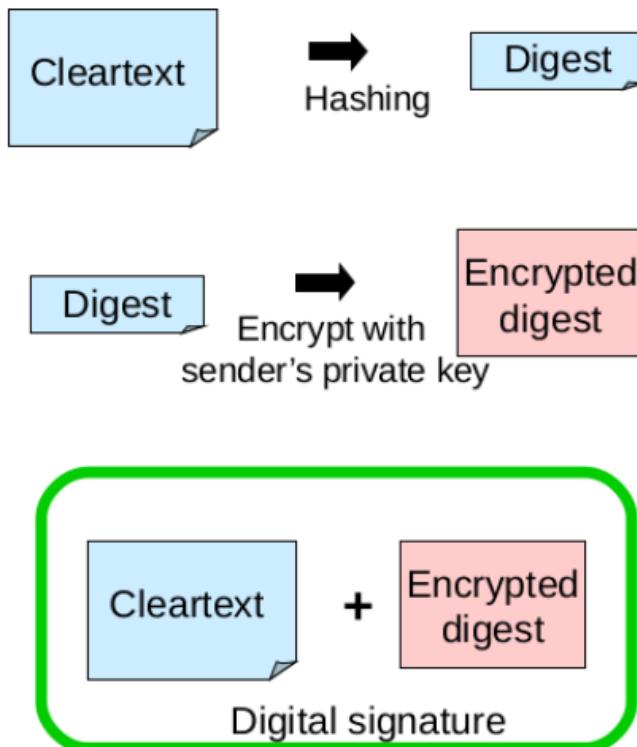
Riassumendo, per fare una firma bisogna fare i seguenti passi:

- Voglio firmare un documento,
- Creo il digest,
- Codifico il digest con la chiave privata del mittente, e metto insieme la coppia (la firma digitale, e quella coppia).

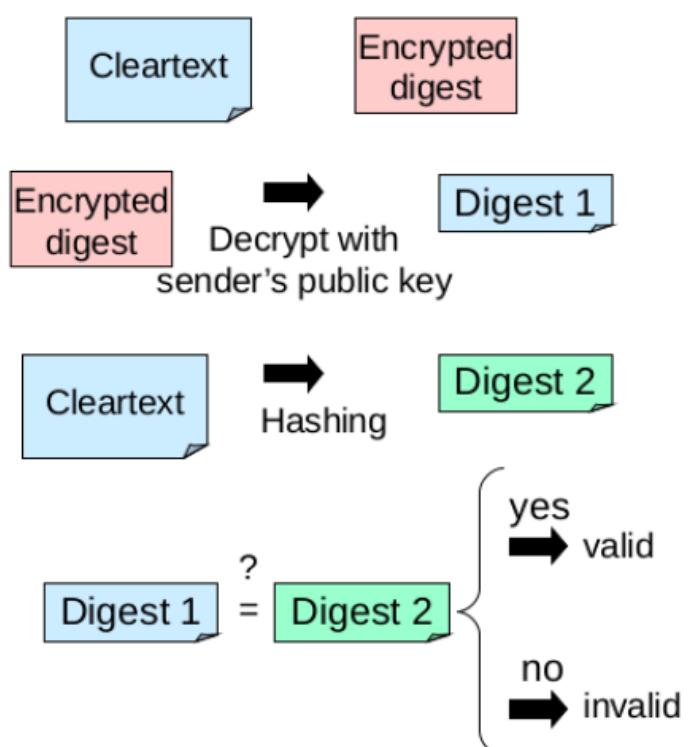
Verificare la firma vuol dire:

- Separare le due componenti,
- Decodificare il digest ottenendone l'hash,
- Fare l'hash dalla parte in chiaro e vedere se i due hash sono uguali.
- Se sono uguali allora la verifica ha esito positivo, altrimenti no.

## Creare una firma



## Verificare una firma



## Protocolli basilari per l'integrità

Consideriamo solo il caso asimmetrico. Per questo caso basta fare la firma digitale (otteniamo pure autenticazione). I prerequisiti sono sempre quelli:

- **Prerequisto 1:** A abbia una chiave privata valida;
- **Prerequisto 2:** B possa verificare che  $k_a$  è di A

$$A \rightarrow B : sign_A(m)$$

Possiamo mettere assieme le proprietà di segretezza, autenticazione e integrità? **La firma digitale non protegge il documento per segretezza**, perché se firmiamo tutti potranno leggere e addirittura tutti possono verificare la firma.

## Combinare tutte le proprietà di livello 1

Segretezza del messaggio  $m$  per  $A$  e  $B$ , autenticazione di  $A$  con  $B$ , integrità di  $m$  nella trasmissione da  $A$  a  $B$

- Crittografia simmetrica, omesso
- Crittografia asimmetrica
  - **Prerequisto1:** B abbia una chiave privata valida
  - **Prerequisto2:** A possa verificare che  $k_b$  è di B
  - **Prerequisto3:** A abbia una chiave privata valida
  - **Prerequisto4:** B possa verificare che  $k_a$  è di A

$$A \rightarrow B : sign_A(m_{k_b})$$

## Facciamo il punto

- Tutti i protocolli basilari visti sono vulnerabili a replay attack!
- Non usano alcun meccanismo di freshness!
- Prima di vedere protocolli più evoluti, manteniamo due vecchie promesse
  1. Crittografia simmetrica: condivisione segreto iniziale
  2. Crittografia asimmetrica: certificazione
- Lo faremo con appositi protocolli!
  1. Crittografia simmetrica: **Diffie-Hellmann (DH)**
  2. Crittografia asimmetrica: **Public-Key Infrastructure (PKI)**

## Protocollo Diffie-Hellmann

Il protocollo Diffie-Hellmann si costruisce nel seguente modo:

- A e B concordano due parametri pubblici  $\alpha$  e  $\beta$ .
- A si costruisce un numero privato  $x_a$ .
- B si costruisce un numero privato  $x_b$ .

$x_a$  e  $x_b$  sono i segreti rispettivamente di A e di B e non li condividono a nessuno.

- A, utilizzando le proprietà delle potenze, calcola

$$Y_a = \alpha^{x_a} \text{ mod } \beta.$$

- Anche B fa lo stesso calcolo

$$Y_b = \alpha^{x_b} \text{ mod } \beta.$$

A e B possono entrambi fare questa operazione perché  $\alpha$  e  $\beta$  sono pubblici.

Quindi  $Y_a$  e  $Y_b$  sono entrambi il risultato dell'operazione di esponenziazione.

- A e B si scambiano  $Y_a$  e  $Y_b$ .

$$1. A \rightarrow B : Y_a$$

$$2. B \rightarrow A : Y_b$$

- Alla ricezione di  $Y_a$ , B l'unica cosa che può fare è prendere il suo segreto,  $x_b$ , e usarlo come esponente per il messaggio che ha ricevuto:

$$Y_a^{x_b} \text{ mod } \beta$$

- Alla ricezione di  $Y_b$ , A l'unica cosa che può fare è prendere il suo segreto,  $x_a$ , e usarlo come esponente per il messaggio che ha ricevuto:

$$Y_b^{x_a} \text{ mod } \beta$$

In questo modo termina il protocollo Diffie-Hellmann.

Questo protocollo, risolve il problema della crittografia simmetrica, ovvero, in questo modo A e B condivideranno una chiave.

**Ma qual'è la chiave che A e B condividono?**

$\alpha$  e  $\beta$  li condividono, ma queste li condividono con tutto il mondo, quindi non possono essere loro le chiavi condivise.

Per le proprietà delle potenze, le ultime due operazioni fatte da A e B sono uguali:

$$Y_a^{x_b} \text{ mod } \beta = Y_b^{x_a} \text{ mod } \beta (*)$$

Perché:

$$Y_a = \alpha^{x_a} \text{ mod } \beta \quad \text{e} \quad Y_b = \alpha^{x_b} \text{ mod } \beta$$

Sostituendoli nella (\*) otteniamo

$$(\alpha^{x_a})^{x_b} \text{ mod } \beta = (\alpha^{x_b})^{x_a} \text{ mod } \beta$$

E per le proprietà delle potenze

$$(\alpha^{x_a})^{x_b} = (\alpha^{x_b})^{x_a} \rightarrow \alpha^{x_a x_b} = \alpha^{x_b x_a}$$

Sia

$$k_{ab} = Y_b^{x_a} \text{ mod } \beta$$

Quindi  $k_{ab}$  è la chiave che stiamo cercando. Questa chiave è segreta? Cioè la condividono solo A e B?

Se l'attaccante è forte con i logaritmi (operazione inversa dell'esponenziazione), potrebbe ottenere i segreti  $x_a$  e  $x_b$  e quindi ricavare la chiave condivisa! In realtà, per poter ottenere i valori  $x_a$  e  $x_b$ , l'attaccante deve risolvere il problema di Matematica discreta del **logaritmo discreto**. Risolvere il problema del logaritmo discreto che a partire, per esempio, da  $Y_b^{x_a}$ , rivelerebbe l'esponente, è un problema intrattabile (computazionalmente intrattabile). Pertanto gli esponenti rimangono protetti, e quindi la chiave condivisa è segreta.

In questo protocollo non c'è uno schema di autenticazione. Quindi laddove non c'è un sistema di autenticazione ci può essere un attaccante **man in the middle**. Vuol dire che, siccome non c'è autenticazione, l'attaccante man in the middle, riuscirà a sfacciarsi per l'altro. Sfacciarsi nel senso che, ad esempio, fa credere al ricevente B, che sta parlando con A, mentre invece sta parlando con l'attaccante. Questo attacco si può fare nel seguente modo:

- L'attaccante C si inventa il suo segreto  $X_c$  e anche lui calcola  $Y_c = \alpha^{x_c} \text{ mod } \beta$ .
- A manda a B il proprio  $Y_a$ , ma C si mette in mezzo e lo altera con il proprio  $Y_c$ .
- Siccome non c'è nessun meccanismo di autenticazione, allora, B accetta quello che riceve, ovvero  $Y_c$ .

Quindi, utilizzando la sintassi dei messaggi crittografici, abbiamo:

1.  $A \rightarrow B : Y_a$  (**bloccato da C**)
- 1'**.  $C(A) \rightarrow B : Y_c$
2.  $B \rightarrow A : Y_b$  (**bloccato da C**)
- 2'**.  $C(B) \rightarrow A : Y_c$

Con  $C(A)$  significa che C impersona A. L'attaccante riesce a dare sia ad A che a B il proprio  $Y_c$ . A e B, che non hanno potuto notare alcun problema di autenticazione, sono programmati per prendere questo messaggio ed elevarlo a potenza. Quindi A e B faranno questo:

Alla ricezione di 2', A calcola  $Y_c^{x_a} \text{ mod } \beta$   
 Alla ricezione di 1', B calcola  $Y_c^{x_b} \text{ mod } \beta$

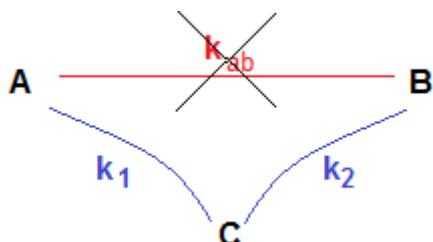
Questi due ultimi numeri generati non sono uguali! Questo perché uno ha esponente  $x_a$  e  $x_c$  e l'altro avrà  $x_c$  e  $x_b$ . Quindi come prima cosa, cade il fatto che i due agenti condividano un segreto, e non solo. Ciò che A ha costruito a modi chiave, sperabilmente condivisa con B, l'attaccante lo può costruire e ciò che B ha costruito a modi chiave, sperabilmente condivisa con A, l'attaccante lo può costruire. Infatti:

- A si è costruito qualcosa, che chiamiamo  $k_1$  ( $k_1 = Y_c^{x_a} \text{ mod } \beta$ )

- **B** si è costruito qualcosa, che chiamiamo  $k_2$  ( $k_2 = Y_c^{x_b} \bmod \beta$ )

Per le proprietà delle potenze  $k_1 \neq k_2$  (quindi non condividono una chiave). C può ricavare sia  $k_1$  che  $k_2$  sfruttando le proprietà delle potenze; cioè  $Y_a$  che ha intercettato, proveniente da A, lo eleva al proprio  $X_c$  e per le proprietà delle potenze ottiene  $k_1$ ; e  $Y_b$  che ha intercettato, proveniente da B, lo eleva al proprio  $X_c$  e per le proprietà delle potenze ottiene  $k_2$ .

Lo schema è drammatico!



Quindi **A** e **B** speravano di condividere una chiave  $k_{ab}$ , per mezzo del protocollo Diffie-Hellmann. Ma niente di tutto ciò accade! **A** condividerà con **C** una chiave  $k_1$  e **B** condivide con **C** una chiave  $k_2$ . **A** manda traffico codificato con  $k_1$  che ritiene sia riconosciuto da **B**, e quindi darà i suoi segreti più intimi a **C** (che potrà intercettarli e decodificarli) e inoltre **C** potrebbe alterare i messaggi di **A** e rispondere lui a **B**. Quindi **C** si spaccia per ambedue gli interlocutori.

Un attacco di questo tipo si chiama **attacco man-in-the-middle**.

**Nota:** la coppia di agenti condividono la chiave e non più di due agenti la condividono.

## RSA Key Exchange

Questo protocollo è ibrido, nel senso che usa sia la crittografia simmetrica che quella asimmetrica. In particolare viene usata la crittografia asimmetrica per fare l'encryption, e poi viene trasportato in questo oggetto, in questo crittotesto, una chiave simmetrica  $k_{ab}$ .

$$1. A \rightarrow B : \{k_{ab}\}_{k_b}$$

Supponiamo che A voglia condividere un segreto con B. A manda il segreto  $k_{ab}$ , codificato con la chiave pubblica del destinatario,  $k_b$ , e grazie alla crittografia asimmetrica, questo segreto viaggerà in maniera confidenziale. Ricevuto questo crittotesto, B (e solo B può farlo), leggerà il segreto inventato da A.

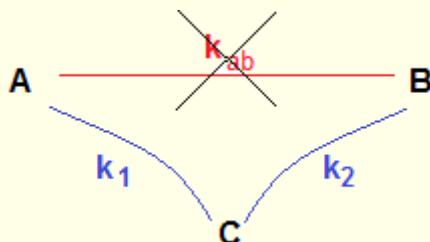
Questo schema si chiama **scambio RSA**, e il crittotesto specifico spesso viene chiamato **busta digitale**.

### **Osservazione**

La chiave di sessione  $k_{ab}$ , viene calcolata tramite Diffie-Hellmann o in modo random?

Diffie-Hellmann ha il limite di autenticazione, allora, per questo protocollo si usa crittografia asimmetrica. Quindi questo scambio RSA dovrebbe proteggere la chiave. Ma nei protocolli base per la sicurezza, quando dovevamo autenticare il messaggio del mittente, utilizzavamo crittografia con la chiave del mittente. Però qua non si vede la chiave del mittente. Ma non serve solo autenticazione, ma anche segretezza, perché lo scopo è trasportare questa chiave in maniera segreta. Ma il limite di Diffie-Hellmann, per cui c'era l'attacco **man-in-the-middle**, era la mancanza di autenticazione. Questo significa che bisogna arricchire Diffie-Hellmann, sia con autenticazione che con segretezza. Quindi: codifica chiave privata del mittente, codifica chiave pubblica del destinatario, affinché Diffie-Hellmann funzioni bene. Ma nello scambio RSA visto prima, non si vede niente di tutto ciò. Quindi, questo scambio RSA, per come è fatto, è vero che usa crittografia asimmetrica, ma non la usa per autenticazione e segretezza, la usa solo per segretezza. Che possiamo dedurre se la chiave di sessione fosse fatta da Diffie-Hellmann?

Supponiamo che A e B si scambiano una chiave  $k_{ab}$  per mezzo di Diffie-Hellmann. Se l'attaccante è “sveglio”, si verifica quello spiegato nel caso di un attacco:



se il protocollo prevedesse che  $Y_a$  e  $Y_b$  siano mandate codificate con la chiave pubblica del destinatario, quest'attacco funzionerebbe ancora? Cioè, quello è l'attacco e immaginiamo che il protocollo Diffie-Hellmann sia trasformato con il concetto di scambio RSA, e che ogni messaggio sia codificato con la chiave pubblica del destinatario. Questo attacco, in queste condizioni, può funzionare? Le modifiche che vengono fatte sono queste:

- |   |                                |
|---|--------------------------------|
| 1. $A \rightarrow B : Y_a$ ( <i>bloccato da C</i> ) | 1'. $C(A) \rightarrow B : Y_c$ |
| 2. $B \rightarrow A : Y_b$ ( <i>bloccato da C</i> ) | 2'. $C(B) \rightarrow A : Y_c$ |

→  
Modificando il  
protocollo

- |                                      |  |
|--------------------------------------|--|
| 1. $A \rightarrow B : \{Y_a\}_{k_b}$ | 1'. $C(A) \rightarrow B : \{Y_c\}_{k_b}$ |
| 2. $B \rightarrow A : \{Y_b\}_{k_a}$ | 2'. $C(B) \rightarrow A : \{Y_c\}_{k_a}$ |

Nel punto 1, A manda a B la chiave  $Y_a$ , ma questa volta, a differenza di quanto detto nel protocollo originario di Diffie-Hellmann,  $Y_a$  viene codificata con la chiave pubblica di B. Se C impersona A quest'attacco non permane, perché poiché questo messaggio è fatto applicando Diffie-Hellmann con l'aggiunta dell'encryption con la chiave pubblica del destinatario, l'attaccante potrebbe prendere  $\{Y_a\}_{k_b}$ , eliminarlo, costruirsi  $\{Y_c\}_{k_b}$  e inviarlo a B. In modo analogo C farà con B verso A (punti 2 e 2'). Quindi A e B calcoleranno le loro  $k_1$  e  $k_2$ , con  $k_1 \neq k_2$  ( $k_1 \neq k_2$  perchè A e B berranno  $Y_c$  anzichè  $Y_a$  e  $Y_b$  rispettivamente). Però l'altra parte dell'attacco, in cui C dovrebbe potersi calcolare  $k_1$  e  $k_2$  non funziona, perché, per poter funzionare, C dovrebbe riuscire a prendere  $Y_a$  e poi  $Y_b$ , i quali invece sono protetti dall'encryption.

Quindi anche con la modifica, la prima parte dell'attacco funziona, perché la codifica non è una codifica di autenticazione, ma è una codifica di confidenzialità (la parte uno è proprio un attacco di autenticazione, perché C si impersona sia ad A che a B). Nella seconda parte, C deve farsi i calcoli, ma per farli gli serve  $Y_a$  e  $Y_b$ . Allora qua C viene fregato, perché non può andare ad apportare modifiche di segretezza, perché  $Y_a$  e  $Y_b$  sono protetti.

Questo è un caso in cui, pur avendo capito che il limite di Diffie-Hellmann era l'autenticazione, fare una modifica per confidenzialità, ci fa vedere che l'attacco non funziona lo stesso. Questo vuol dire che il limite di Diffie-Hellmann è anche un limite di confidenzialità, perché si è visto che ci sono due attacchi: la prima è la violazione di autenticazione, quando C impersona A o B e l'altra è la violazione di segretezza, quando C legge  $Y_a$  e  $Y_b$ . Ecco perché una misura o di segretezza o di autenticazione risulta codificata. Lo stesso attacco si poteva risolvere se i due messaggi Diffie-Hellmann fossero stati codificati con la chiave privata del mittente, perché risolvevamo la prima parte di attacco anziché la seconda.

**Conclusione: possiamo usare sia Diffie-Hellmann che una chiave random.**

## Certificazione

Il problema della crittografia asimmetrica è (ogni volta che codifichiamo con una chiave pubblica), quello di sapere a chi appartenga quella chiave. Se sbagliamo in questo punto, c'è un attacco di segretezza sulla busta digitale. Questo problema si risolve con la **certificazione**.

Un esempio di certificazione è la nostra carta di identità: la nostra carta di identità associa un viso a una anagrafica. Tutti i certificati fanno associazioni di set di informazioni. Ad esempio se l'utente A ha preso 110, c'è A e c'è 110; queste due informazioni stanno assieme.

**Tutti i certificati suggellano associazioni.**

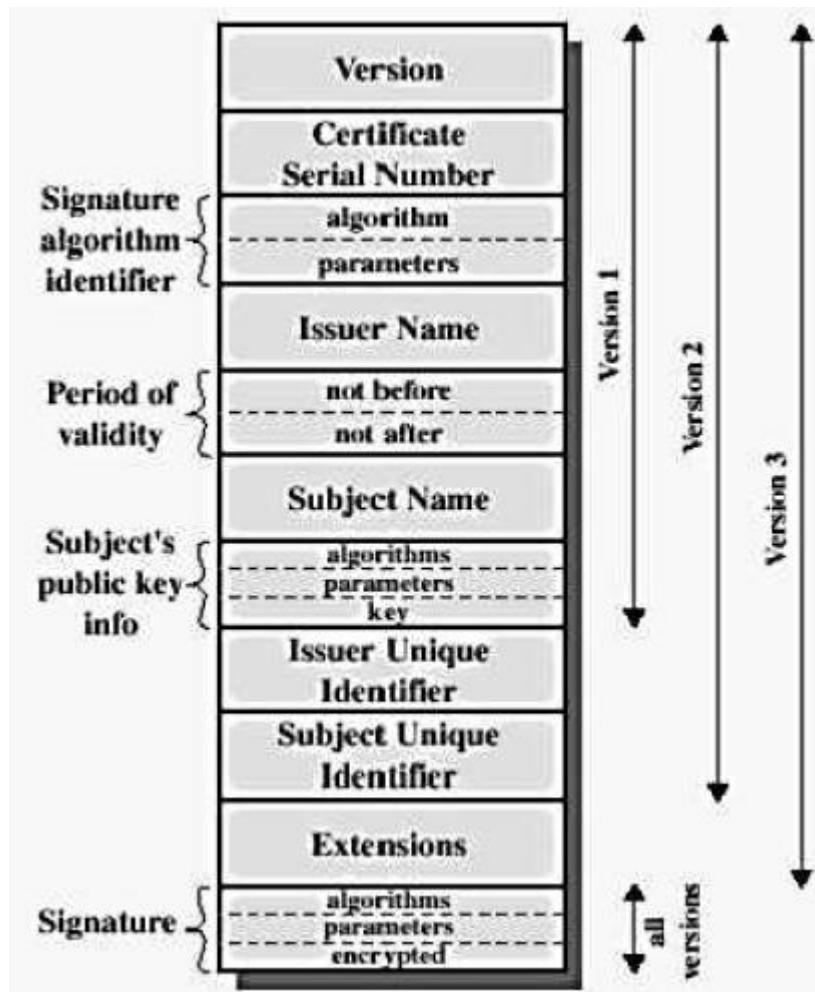
Il certificato nel mondo digitale è un **certificato digitale**. Un certificato è **firmato**. Se non fosse firmato da una autorità potremmo ad esempio dire che *Franco* è il presidente della repubblica, firmato *Pippo*. Chi firma l'associazione deve avere

l'autorevolezza, l'autorità per fare questa firma. Nel caso digitale, chi ha questa autorità si chiama **autorità di certificazione**.

I consorzi di banche costruiscono le autorità di certificazione. Essi hanno l'autorevolezza per sugellare l'autenticazione fra identità e stringhe (la chiave pubblica è una stringa).

Il formato standard del certificato si chiama **X.509**.

## Certificato X.509



Un certificato digitale è l'associazione tra il nome di un agente ( $A$ ) e una stringa di bit (abbiamo altre informazioni ma dal punto di vista notazionale possiamo "scordarli"). Quest'ultimo sarà un certificato, ovviamente, se viene codificato. Più che codificarlo deve essere firmato, quindi il certificato è garantito per autenticazione e integrità, con la chiave privata ( $k_A$ ) di una autorità di certificazione:

$$\{A, k_A, \dots\}_{k_{CA}^{-1}}$$

Quest'ultima scritta è una versione semplificata notazionalmente dell'immagine. I certificati hanno una scadenza.

## Public-Key Infrastructure (PKI)

Nella figura precedente, si vede che, il campo fondamentale è quello che sta al centro; informazioni sulla chiave pubblica. La chiave pubblica significa che è visibile a tutti. In particolare, nel certificato, sta a indicare che essa appartiene a qualcuno. Vuol dire che quel qualcuno conosce l'altra metà della chiave, ovvero la chiave privata.

Quanto descritto prima è come funziona la certificazione. Significa che se A vuole mandare una chiave di sessione a B, deve prendere la chiave di sessione, codificarla con la chiave pubblica di B e mandarla a B. In questo modo A e B avranno una chiave condivisa. Quindi solo A e B avranno questa chiave ma solo se A ha la chiave pubblica di B e il relativo certificato. Ma un certificato può risolvere questo problema? C'è differenza tra certificato, che è un oggetto (il quale potrebbe stare in tasca, chiuso in un cassetto), e un protocollo. Ad esempio: se ci iscriviamo in palestra, dobbiamo dare il certificato medico, e diciamo che ce lo abbiamo, ma è a casa dentro il cassetto. Da qui me deriva che, non è il concetto che risolve il problema. È il protocollo che usa l'oggetto che risolve il problema. Questo protocollo si chiama **PKI**.

Quindi:

1. Per comunicare con A, a B serve il certificato di A
2. Schematizzando, esso ha forma  $\{ \dots A \dots k_a \dots \}_{k_{ca}^{-1}}$
3. Quindi A ha bisogno del certificato della CA
4. A sua volta, questo è firmato da un'autorità superiore, ed ha forma  $\{ \dots CA \dots k_{ca} \dots \}_{k_{ca(n-1)}^{-1}}$
5. Pertanto, ciascuna autorità è certificata da una superiore, fino alla **root certification authority (RCA)** o **primary certification authority**

Una **infrastruttura a chiave pubblica** o **sistema di certificazione** consiste nella gerarchia di autorità e nelle tecnologie che esse usano

Dal punto 4, si vede che il certificato X deve essere firmato da una autorità superiore X+1; X+1 da un'altra autorità superiore X+2, e così via. Prima o poi questo gioco si deve fermare. Questo accade, quando la firma dell'autorità superiore viene riconosciuta da tutti. Ad esempio, potrebbe capitare di presentare una piccola catena di certificati; il datore di lavoro, ricevuto il primo certificato dove viene specificato che è stata data la materia Internet Security, vuole un certificato dal presidente del corso di Laurea per verificare che:

- Abbiamo dato Internet Security con Giampaolo Bella
- Che Giampaolo Bella è il docente del corso.

- Che la firma di Giampaolo Bella sia quello scarabocchio presentato al datore di lavoro.

Ma questo datore di lavoro vuole un altro certificato per verificare che la firma presente nel secondo certificato sia del presidente del corso di laurea. Questo processo prima i poi deve finire. Finirà quando, ad esempio, otterremo un certificato con la firma del presidente della repubblica. Questo perché, dobbiamo arrivare all'assunzione che la firma del presidente della repubblica noi tutti la riconosciamo, altrimenti questo gioco non finirà mai.

Tutto questo procedimento viene fatto dai Browser ogni qualvolta scriviamo https. Quindi tutti i nostri Browser conoscono la chiave pubblica di tante root (non c'è una root più importante o meno importante).

## Chain of Trust

Da quanto detto prima, nasce il concetto di **catena di fiducia**.

La **catena di fiducia** di  $A$  è la lista completa dei certificati (fino a quello di root) che permettono, insieme alla chiave pubblica della RCA, di verificare il certificato di  $A$

Indichiamo con  **$CA(n)$**  il **certificato foglia**, firmato da una autorità di livello ultimo,  $n$ .

$$\{ \dots A \dots k_a \dots \}_{k_{ca(n)}^{-1}}$$

Iterando il problema, il mio browser per risolvere questo certificato, si procura la chiave pubblica di questa autorità di certificazione che è esattamente la similitudine fatta prima. Quindi mi serve un oggetto fatto così:

$$\{ \dots CA(n) \dots k_{ca(n)} \dots \}_{k_{ca(n-1)}^{-1}}$$

**$CA(n)$**  ha la chiave pubblica  $k_{ca(n)}$ , firmato da qualcuno di "più grosso":  $k_{ca(n-1)}^{-1}$ . Questo procedimento continua. Arriverò al certificato di livello 1

$$\{ \dots CA(1) \dots k_{ca(1)} \dots \}_{k_{rca}^{-1}}$$

firmato dall'autorità di certificazione di livello zero, ovvero, **RCA**. L'ultimo certificato è il certificato di root:

$$\{ \dots RCA \dots k_{rca} \dots \}_{k_{rca}^{-1}}$$

**Domanda:** Qual è la differenza sintattica tra certificato generico e certificato di root?

**Risposta:** In tutti i certificati, la chiave che viene usata per firmare il certificato è di livello superiore a quella certificata, tranne nel caso di root. **Il certificato di root è l'unico autofirmato.**

## Segretezza vs. Trust

Trust vuol dire fiducia.

Supponiamo che l'attaccante violi un'autorità di certificazione.

Ricorda che, un'autorità di certificazione sta in un albero, l'albero della certificazione, nella PKI.

Un'autorità di certificazione è compromessa quando la sua chiave privata la rilevata al nemico. Ad esempio: "lauree false"; il rettore viene pagato e firma che determinate entità hanno la laurea in Informatica. Lo scopo della certificazione ha a che fare con la segretezza o con la fiducia? **La certificazione si basa sulla fiducia.** Se un attaccante compromette un'autorità di certificazione, non automaticamente si impossesserebbe di altre chiavi private.

Supponiamo che l'attaccante compromette l'autorità  $i$ -esima; **CA( $i$ )**. Al di sotto c'è un sottoalbero. Se l'attaccante ha corrotto l'autorità  $i$ -esima non ha corrotto tutto il sottoalbero, cioè non si impossessa delle chiavi private delle autorità che stanno sotto. Questo perchè non necessariamente ha corrotto pure le autorità del sottoalbero. Riprendendo l'esempio del rettore, se esso viene corrotto, non necessariamente un professore è stato corrotto; quindi magari il rettore fa certificati falsi se è corrotto, ma questo non vuol dire che possiamo fare certificati falsi con la chiave privata di un determinato professore X. Quindi solo l'autorità  $i$ -esima ha perso la segretezza e non è detto che questo accade con le autorità inferiori. È questo il nesso tra segretezza e fiducia. **La violazione di segretezza non si propaga ai livelli sottostanti.**

Se l'attaccante ha corrotto l'autorità  $i$ -esima, automaticamente tutti perderebbero fiducia in qualunque chiave pubblica la cui certificazione necessiti della chiave pubblica della **CA( $i$ )** perché l'attaccante potrebbe creare delle false **CA( $i + 1$ ) ... CA( $n$ )** aventi chiavi pubbliche con certificati falsi. Quindi ad esempio, il rettore vuole certificare che il suo nipotino piccolo è docente di Programmazione 2. Questo lo può fare ed inoltre avrà delle implicazioni sul certificato del docente X, perché sarà il rettore a confermare il tutto. Allora dobbiamo credere che il docente X sia il docente di Sicurezza? Anch'esso passerebbe per una autorità corrotta. Bisognerebbe fare un'analisi temporale e verificare se il docente X è anteriore o posteriore nel momento in cui il rettore è stato corrotto. Una cosa di questo tipo è impossibile.

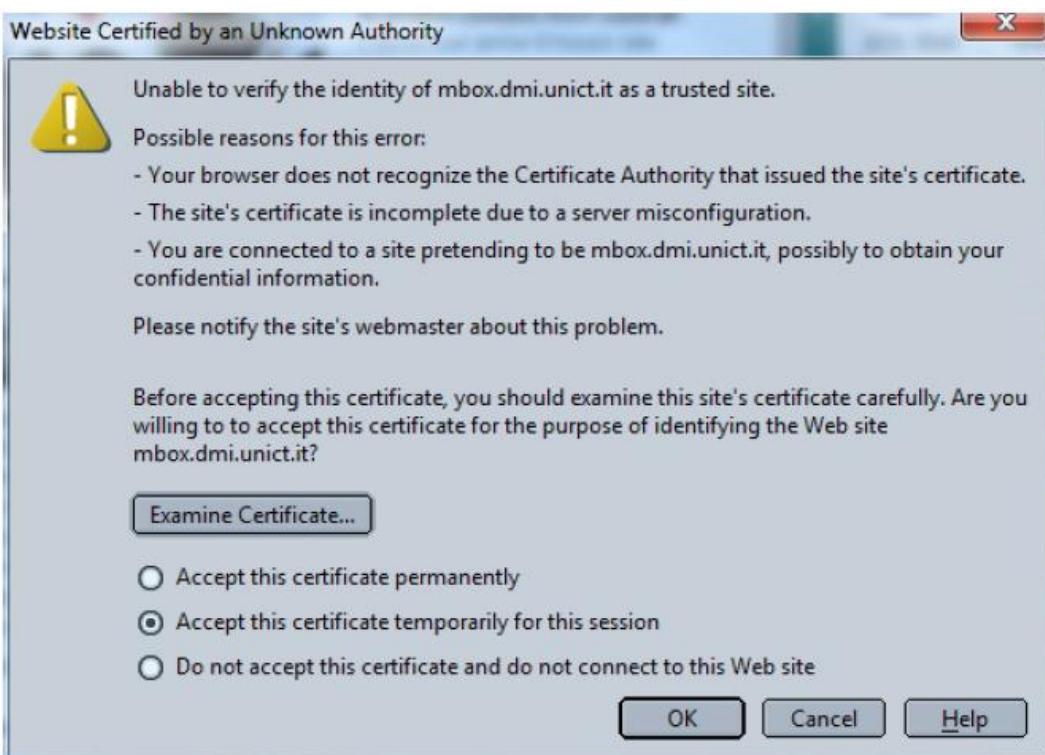
Quindi quello che si è verificato non è un problema di segretezza, ma è un problema di fiducia. E allora anche se la mia chiave segreta è rimasta privata non me ne faccio niente, perché nel momento in cui codificassi con la mia chiave pubblica o con un'altra chiave, i crittotest si perderebbero di fiducia, perché poiché il tutto verrà "verificato" dal rettore ed esso è corrotto, i messaggi perderebbero il loro potere di esprimere fiducia come proveniente dal docente X.

La perdita di segretezza non si propaga; la perdita di trust si propaga verso i livelli inferiori

## Problema: certificati self-issued

I certificati di root sono installati nei Browser. Essi sono una risorsa sensibile. Potremmo dare al browser a bere qualsiasi cosa, perché possiamo noi (o l'attaccante) inserire certificati creati da noi. Dare a bere al browser in maniera legittima significa che l'address bar è verde e il lucchetto è chiuso. In maniera illegittima è al contrario: l'address bar è rossa e il lucchetto è aperto, e al secondo del browser avremo dei messaggi di warning.

Consideriamo i certificati “fatti in casa”. Vediamo quello che è successo un paio di anni fa al DMI. A qualcuno era stato assegnato l'incarico di mettere la posta su https, perché se è su http la password è più debole, viaggia in chiaro. Questo tizio, doveva abilitare questo accesso e dopo alcune operazioni, doveva inserire il certificato del mailbox. Esso può essere o acquistato o generato manualmente. È stato appunto creato un certificato di **mbox.dmi.unict.it**, firmato “centro di calcolo”. Firmato “centro di calcolo” vuol dire che questa era la chiave privata usata per la firma, generata a sua volta per l'entità centro di calcolo. Ogni qualvolta qualcuno andava a leggere la posta, veniva fuori un messaggio di questo genere:



Questo messaggio dipende dal Browser. Le possibili scelte che spuntano sono:

1. Non accettare questo certificato e quindi non accedere al sito
2. Accetta questo certificato temporaneamente per questa sessione
3. Accetta questo certificato permanentemente

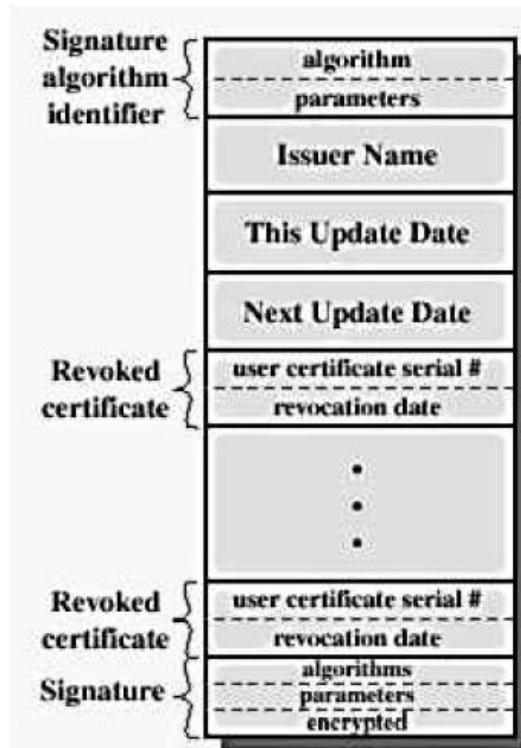
Il punto 1 non lo consideriamo nemmeno. Se viene accettato solo per questa sessione, l'indomani riapparirà questo messaggio. Dal punto di vista della sicurezza quale tra i punti 1 e 2 conviene accettare? Se accettiamo per sempre il certificato e questo è stato dato dall'attaccante saremmo fregati per sempre.

Se invece il certificato fosse stato comprato e arriva un messaggio del genere la situazione diventa abbastanza critica, perché questo è un router malevolo, che non darà la pagina di mbox reale/certificata. L'utente imbecille, prima che capisca cosa stia succedendo, avrà inserito la sua password della posta e l'attaccante avrà ottenuto il suo scopo (avrà scoperto la password di posta).

Quindi dobbiamo accettare certificati, solo quando siamo sicuri di non essere sotto attacco.

## Certificate Revocation List (CRL)

Una **lista di revoca dei certificati (CRL)** è tanto sensibile quanto un certificato. Per fare una CRL ci vuole tanta autorità, almeno quanto ce ne voglia per fare un certificato. È chiaro che anche revocare implica una autorevolezza. La CRL ha anch'essa in basso la parte della firma, una lista di certificati revocati, ha identificativi della firma, chi fa la CRL e le date della CRL.



Revocare un certificato significa invalidarlo prima della sua scadenza. Questo avviene se la chiave privata, corrispondente alla chiave pubblica che il certificato racchiude, venga in qualche modo smarrita, violata, compromessa. Questa operazione la può fare lo stesso proprietario o l'autorità di certificazione. Se il certificato viene revocato, il destinatario di qualunque messaggio, fatto con crittografia asimmetrica, deve sapere se il certificato è revocato, perché se questo certificato è revocato possibilmente vuol dire che la chiave privata  $k_a^{-1}$  è compromessa, ovvero  $\exists C : C \text{ conosca } k_a^{-1} \text{ e } C \neq A$ , pertanto quell'oggetto potrebbe non più necessariamente venire da A ma da un certo C.

Se il certificato è scaduto, l'autorità di certificazione dice che non ha più niente a che vedere con quel certificato, perché l'autorità di certificazione ha venduto la fiducia (trust) fino a giorno tot. Dal giorno dopo la scadenza, tot+1, l'autorità di certificazione non farà più controlli.

**Esempio:**

Se compriamo un biglietto dall'Alitalia, il browser ha bisogno del certificato di *Alitalia.com*. Se il certificato è malevolo, il browser potrebbe indirizzarmi su *AlitaliaB.com* ed esso potrebbe essere un sito malevolo. Il browser, segnalerà una cosa di questo tipo, ma se l'utente continuerà e quindi decide di accedere al sito, finirà su *AlitaliaB.com*, si troverà inguaiato.

# **PROTOCOLLI DI SICUREZZA STORICI**

## Protocollo di sicurezza – esempio 1: *Needham-Schroder asimmetrico*

Consideriamo il seguente protocollo “giocattolo”:



Questo protocollo è stato inventato da **Needham-Schroder** nel 1978. È uno dei primi utilizzi della crittografia asimmetrica.

Tra Alice e Bob c'è un attaccante DY. Lo scopo di questo protocollo è che Alice autentica Bob e viceversa, solo che in mezzo c'è questa rete insicura. C'è crittografia asimmetrica, c'è una PKI con crittografia perfetta; quindi tutti sanno la chiave pubblica giusta da usare.

Alice e Bob in qualche modo si devono autenticare. Dall'immagine si vede che viene usato uno schema simile allo schema base dell'autenticazione, ovvero Alice codifica il messaggio con la chiave pubblica di Bob. In realtà il protocollo base funzionava in questo modo: Alice mandava il messaggio codificato con una sua chiave privata; Bob lo riceve, lo apre, usa la chiave giusta e autentica Alice grazie alla certificazione. In sostanza, bastava un solo messaggio. Ma vogliamo inserire la freshness tramite nonce. Quindi se dobbiamo autenticare Bob con Alice, è Alice che prima deve inventarsi la nonce e poi riaverla indietro da Bob.

Alice si inventa la nonce e la manda a Bob, codificata con la chiave pubblica di Bob; questo perché Alice sta cercando di autenticare Bob.

Alla ricezione del messaggio 1, Bob è l'unico che lo può decodificare. Bob tira fuori la nonce e la manda indietro ad Alice. Il protocollo base si fermava con il messaggio 2 che vedeva la nonce viaggiare in chiaro da Bob ad Alice, perché ormai non era più un problema, quindi non c'era bisogno di codificare la nonce con la chiave pubblica del destinatario (Alice), dato che il destinatario sarebbe stato solo lui in grado di decodificare il messaggio. Solo che Bob chiede le stesse garanzie ad Alice. Quindi Bob si crea una nonce, e la manda codificata ad Alice (codificata con la chiave pubblica di Alice).

Questo protocollo lo potevamo scindere. Potevamo fare che nel messaggio 2 Bob manda ad Alice  $N_a$  e nel messaggio 3 Bob manda ad Alice  $N_b$  codificata con la chiave pubblica di Alice:

$$(*) \begin{cases} 2. B \rightarrow A : N_a \\ 3. B \rightarrow A : \{N_b\}_{k_{Alice}} \end{cases}$$

Nella figura, però, il secondo messaggio lo possiamo considerare come

$$(**) \quad \{N_a, N_b\}_{k_{Alice}}$$

Quest'ultimo messaggio è la combinazione dell'(\*); quindi in questo modo viene restituito  $N_a$  ad Alice e l'altro serve ad esprimere il challenge di Bob. In sostanza per semplicità, possiamo usare la notazione (\*\*).

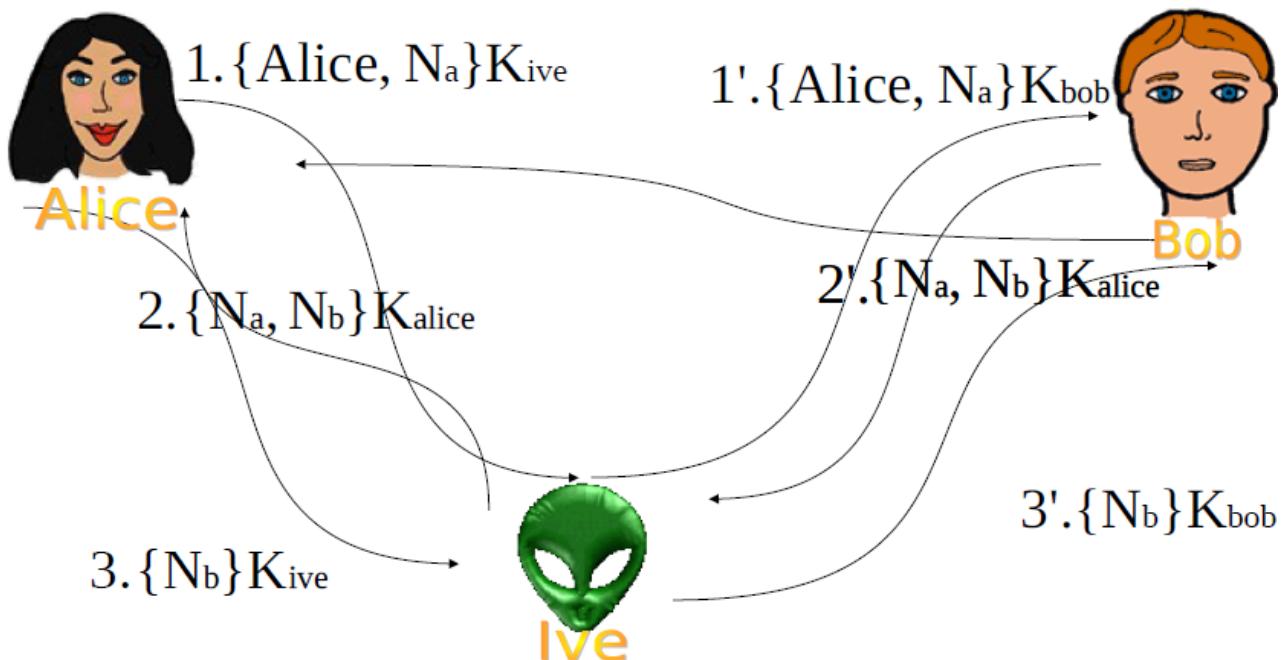
**Non c'è un problema di replay perché c'è freshness:** Alice può annotarsi il momento in cui si inventa  $N_a$ , e messaggi più vecchi del momento dell'invenzione di  $N_a$ , Alice non ne accetterà (ciò che Alice riceve dopo aver creato  $N_a$  viene accettato).

Quindi il messaggio 2 serve ad autenticare Alice con Bob, ma nello stesso tempo trasporta anche il challenge di **B** ad **A**. Alice è l'unica entità che può decodificare il messaggio 2; tira fuori  $N_a$  (capisce che dall'altro lato c'è Bob) e  $N_b$  e rimanda  $N_b$  indietro codificando questo messaggio con la chiave pubblica di Bob. A questo punto Bob autentica Alice (perché Alice è l'unica che avrebbe potuto estrarre  $N_b$  dal messaggio 2). Il messaggio 3 non serve che sia codificato.

Notare che Alice e Bob condividono  $N_a$  e  $N_b$ , le quali sono segrete perché mandate crittografate.

Tutta questa analisi di sicurezza che conferma la bontà di questo protocollo è stata accettata dal 1978 al 1993. **Per 15 anni questo protocollo è stato considerato buono.** Ma nel 1993 qualcuno osservò che **questo protocollo ha almeno uno scenario di attacco**.

Qualcuno inventò lo scenario mostrato nella figura seguente, in cui ci sono due sessioni intrecciate del protocollo (questa affermazione ha senso perché molte volte abbiamo due sessioni HTTPS con due agenti diversi).



Nella prima sessione i protagonisti sono Alice e IVE. Nella seconda sessione i protagonisti sono IVE e Bob. Alice inizia il proprio protocollo con IVE. Inventa la nonce  $N_a$ , prepara il messaggio "**Alice,  $N_a$** " e codifica tutto quanto con la chiave pubblica del destinatario che è  $k_{ive}$ :

$$1. \{Alice, N_a\}_{k_{ive}}$$

Ive riceve questo messaggio 1 (solo lui può decodificarlo). Ma Ive è l'attaccante DY, e di conseguenza non segue le regole. Dopo aver decodificato il messaggio 1, Ive non manderà il messaggio 2, ma con la nonce di Alice,  $N_a$ , fa un replay di questa nonce su una nuova sessione con il povero Bob. Quindi Ive costruisce il messaggio 1':

$$1' : \{Alice, N_a\}_{k_{bob}}$$

Questo messaggio Ive lo può fare. Dal punto di vista sintattico questo messaggio non ha niente di sbagliato, ma c'è un abuso poiché non sta mettendo una nuova nonce.

Bob decodifica il messaggio 1' che riceve. Vede che si tratta di Alice e andrà a costruire un messaggio per Alice. Questo messaggio sarà codificato con la chiave pubblica di Alice:

$$2' : \{N_a, N_b\}_{k_{alice}}$$

Questo messaggio potrebbe andare direttamente ad Alice.

Ive utilizza Alice come **oracolo** (termine usato in crittografia). Questo è possibile farlo perché Alice aveva una sessione con Ive. Quindi Alice, riceve il messaggio 2, lo può decodificare. In questo passaggio riconosce la nonce che aveva creato e quindi Alice conclude che dall'altro lato della connessione c'è Ive. In sostanza quello che accade in questo scenario è questo: Alice autentica Ive attraverso la ricezione del messaggio 2, tira fuori la nonce di Bob e Alice è programmata per codificarla con la chiave pubblica del suo "amico" Ive.

### Notare che:

Non è perché in  $N_b$  c'è come pedice il carattere 'b' che Alice riconosce che questa nonce è stata creata Bob.  $N_b$  è una stringa.

A questo punto, Ive riceverà il messaggio 3 da Alice e può decodificarlo.

$$3. \{N_b\}_{k_{ive}}$$

Dopo questa decodifica Ive si è impossessato della nonce  $N_b$  inventata da Bob per essere condivisa con Alice, ed è così che Ive può completare il misfatto. Prende la nonce  $N_b$  e la codifica con la chiave pubblica di Bob. Bob rivenuto questo messaggio, lo decodifica e riconosce la sua nonce, e poiché precedentemente l'aveva decodificata con la chiave pubblica di Alice è sicuro che dall'altra parte c'è Alice. Quindi Bob è convinto di avere una sessione con Alice, mentre invece ha una sessione con Ive.

Questo è il **prototipale attacco man in the middle**.

Questo attacco ha due sessioni interlacciate. La segretezza di  $N_b$  fallisce con il passo 3, perché questa nonce viene scoperta dall'attaccante. Quindi se prima dicevamo che le nonce erano segrete e quindi si aveva una bella chiave di sessione, ora non lo possiamo più dire, perché l'attaccante con questo schema scopre  $N_a$  e  $N_b$ . Scopre  $N_a$  legittimamente, perché Alice gli e la data, mentre  $N_b$  illegittimamente. Finisce male sia la segretezza che l'autenticazione, malgrado la crittografia sia perfetta.

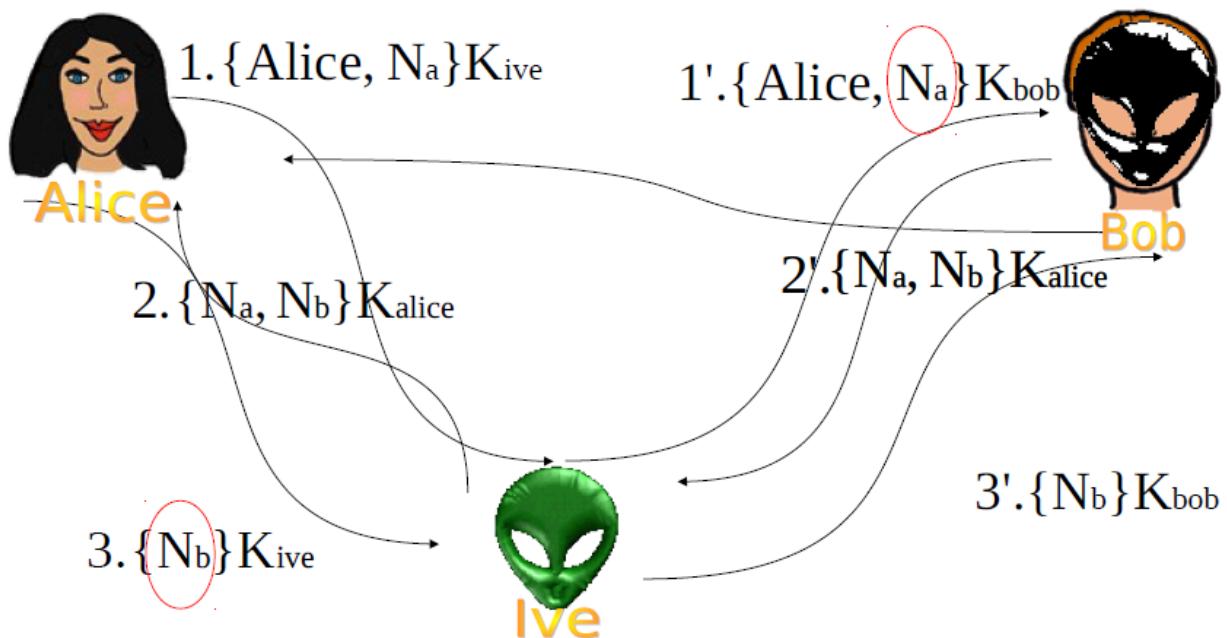
L'attacco esiste in un modello di attaccante DY, che però non è quello per il quale era stato progettato (quando questo protocollo è stato realizzato non c'erano i problemi che ci sono oggi giorno).

Bob pensa di avere una sessione con Alice, ma in realtà è Ive, che si autentica come Alice. Questo potrebbe causare gravi problemi. Consideriamo il seguente messaggio:

*Cara Banca Bob, sono Alice. "Sono Alice" va convalidato, per esempio esibendo le chiavi di sessione. Alice potrebbe chiedere a Bob di trasferire soldi dal suo conto al conto di Ive.*

## Lo stesso attacco studiato in GA

Consideriamo lo stesso protocollo descritto prima e supponiamo di scontrarlo con un **general attacked**.



Consideriamo le parti cerchiate in rosso. Essi rappresentano il vero problema. Il primo ovale (punto 3) è un problema perché è il momento in cui Ive scopre la nonce  $N_b$ , che è inventata da Bob per essere condivisa con Alice. Questo è un attacco. Ma se anche Bob avesse intenzioni malevoli cambierebbe il modello di attaccante. In questa situazione c'è un momento in cui Bob si accorge della nonce  $N_a$ . Questa nonce è stata creata da Alice che la voleva condividere con Ive. Bob non ha idea che questa nonce la conosce sia Alice che Ive.

Questo è un **attacco di replication**. Il tutto è iniziato da Ive, mandando a Bob la nonce creata da Alice. Allora Bob, quando vuole, può abusarla ( $N_a$ ) presso Alice, la quale ignora del fatto che questa coppia di nonce adesso sia condivisa anche con Bob, poiché pensa di condividerla solamente con Ive.

## Protocollo di sicurezza – esempio 2

Il seguente protocollo è basato sulla crittografia simmetrica. Essendo basato su crittografia simmetrica non ci stupisce che venga fuori un server fidato, chiamato **TTP**. TTP (**Trusted Third Party**), vuol dire terza parte di fiducia, possiede un database di chiavi.

Quando parliamo di crittografia simmetrica, quasi sempre ci dobbiamo aspettare un server fidato, perché se ciascuno di noi ha una password, ovvero una chiave segreta, e vogliamo usarla per comunicare a due a due in maniera sicura, possiamo usare Diffie-Hellmann (ma questo alla fine scivola sulla crittografia asimmetrica) o altro. Abbiamo bisogno di un garante fidato che conosce tutto.

1.  $A \rightarrow B : A$
2.  $B \rightarrow A : N_b$
3.  $A \rightarrow B : \{N_b\}K_a$
4.  $B \rightarrow TTP : \{A, \{N_b\}K_a\}K_b$
5.  $TTP \rightarrow B : \{N_b\}K_b$

Alice si vuole autenticare con Bob. L'unico modo che ha Alice per convincere Bob che è veramente Alice è quello di dire “Caro Bob sono Alice”:

**1.  $A \rightarrow B : A$**

Ovviamente questo solo messaggio non può realizzare l'autenticazione perché può essere violabile facilmente. Bob appena riceve il messaggio 1 potrebbe dire “Si direbbe che c'è Alice dall'altra parte”. Bob sfida Alice, inviando una nonce,  $N_b$ , per verificare cosa Alice è in grado di fare con quest'ultima:

**2.  $B \rightarrow A : N_b$**

Alice ha una chiave a lungo termine simmetrica, quindi, se riceve il messaggio 2, l'unica cosa che può fare con la nonce è prenderla e codificarla con la propria chiave a lungo termine.

**3.  $A \rightarrow B : \{N_b\}_{k_a}$**

Se Bob riceve il messaggio 3, lui non può farci niente con questo challenge codificato, perché non conosce la chiave a lungo termine di Alice. Allora Bob quello che può fare è costruire un nuovo messaggio, contenente il messaggio codificato appena ricevuto ( $\{N_b\}_{k_a}$ ) e scrive accanto “Secondo me viene da Alice”. Questo messaggio lo codifica con la propria chiave a lungo termine,  $k_b$ , e lo manda a TTP:

**4.  $B \rightarrow TTP : \{A, \{N_b\}_{k_a}\}_{k_b}$**

Quest'ultimo messaggio lo possiamo leggere così: “*Il challenge codificato da parte di Alice (seconda componente), con affianco (affianco ≡ concatenazione) il nome di Alice che è l'interlocutore atteso/Ipotetico da confermare di Bob; il tutto codificato con la chiave di Bob* (la codifica di Bob serve ad evitare che DY intercetti il messaggio e al posto di Alice metta Charlie)”.

Il senso di questo messaggio è che Bob vuole dire a TTP che ha ricevuto un challenge e che crede che provenga da Alice. Bob vuole sapere da TTP se è veramente Alice. Ma TTP non può dire se quel determinato challenge è codificato da Alice, ma quello che sa TTP è la chiave di Alice. Quindi TTP, pensando che quel challenge sia di Alice, può decodificarlo con la chiave di Alice. Ma per fare questo, bisogna prima controllare che il challenge originale, con il risultato di questa decodifica che solo TTP può fare, corrisponda. Se non corrispondono vuole dire che Alice non era Alice. Dopo tutti questi passaggi, TTP manda a Bob il messaggio decodificato, ovvero,  $N_b$ , codificandolo con la chiave di Bob:

$$5. \text{ TTP} \rightarrow B : \{N_b\}_{k_b}$$

A questo punto Bob, una volta ricevuto il messaggio da TTP e dopo averlo decodificato, può controllare se quel challenge,  $N_b$ , era quello che lui aveva creato. Se questo matching ha esito positivo, Bob autenticherà Alice.

**Anche questo protocollo sembra sicuro!**

## Un attacco su Woo-Lam

La parte iniziale del protocollo descritto prima è “debole”. Se l’attaccante si vuole spacciare per un altro verso Bob, lo può fare.

- 1.  $A \rightarrow B : A$
- 2.  $B \rightarrow A : N_b$
- 3.  $A \rightarrow B : \{N_b\}K_a$
- 4.  $B \rightarrow \text{TTP} : \{A, \{N_b\}K_a\}K_b$
- 5.  $\text{TTP} \rightarrow B : \{N_b\}K_b$

- B vede indietro  $N_b$
- Pertanto autentica l’utente cui l’ha associata, ossia A
- A potrebbe perfino essere off-line
- B non distingue la sessione!

- 1.  $C \rightarrow B : A$
- 1'.  $C \rightarrow B : C$
- 2.  $B \rightarrow A : N_b$
- 2'.  $B \rightarrow C : N'_b$
- 3.  $C \rightarrow B : \{N_b\}K_c$
- 3'.  $C \rightarrow B : \{N'_b\}K_c$
- 4.  $B \rightarrow \text{TTP} : \{A, \{N_b\}K_c\}K_b$
- 4'.  $B \rightarrow \text{TTP} : \{C, \{N'_b\}K_c\}K_b$
- 5.  $\text{TTP} \rightarrow B : \{N_b\}K_b$
- 5'.  $\text{TTP} \rightarrow B : \{N'_b\}K_b$

Nella figura si hanno due sessioni parallele. Le fa ambedue l’attaccante, C. Su una sessione cita se stesso, su l’altra cita A che è offline (messaggio 1 e 1’).

B replica correttamente a ambedue le sessioni e replica con due challenge diversi,  $N_b$  e  $N'_b$  solo che l’attaccante intercetta il traffico dedicato ad A e ci fa quello che vuole (messaggio 2 e 2’). L’attaccante non fa operazioni particolari (non viola la crittografia). Quello che fa è che butta via un challenge,  $N'_b$ .

Successivamente, l'attaccante, codifica il primo challenge,  $N_b$ , con la propria chiave,  $k_b$ , e anche sull'altra sessione manda lo stesso challenge (messaggi 3 e 3'). Questa versione è diversa dal man in the middle, perché quest'ultimo faticava per procurarsi un challenge codificato dalla vittima; per questo la vittima ci voleva. Questa logica invece, non la usa proprio la vittima (A), ma bensì, l'attaccante spaccia su ambedue le sessioni il challenge codificato che la vittima (B) ha associato ad A.

B riceve i challenge e li manda a TTP (messaggio 4 e 4'). Su una sessione B pensa che l'interlocutore atteso sia A, sull'altra pensa che sia l'attaccante C.

A questo punto, B manda a TTP il challenge. Nella prima sessione, TTP

- riceve il messaggio 4,
- lo decodifica,
- tira fuori la seconda componente,
- legge la prima parte della seconda componente, A,
- cerca la chiave di A e decodifica la seconda parte della seconda componente con la chiave di A,
- tirando fuori un certo  $N''_b$ .

Analogamente, nella seconda sessione, TTP

- riceve il messaggio 4',
- lo decodifica,
- tira fuori la seconda componente,
- legge la prima parte della seconda componente, C,
- cerca la chiave di C e decodifica la seconda parte della seconda componente con la chiave di C,
- tira fuori  $N_b$

TTP rimanda indietro i risultati ottenuti (messaggio 5 e 5'). Sulla prima sessione, B, vede un  $N''_b$  che non riconosce; quindi fa abort. Sulla seconda sessione, vede  $N_b$  che riconosce, ovvero quella  $N_b$  che ha associato ad A e quindi autentica A.

L'attaccante all'inizio deve citare se stesso, perché l'unico challenge che può codificare per primo è con la sua chiave, perché quello stesso C sarà proprio la C che andrà nel messaggio 4'. Quindi la decodifica andrà a buon fine proprio perché lui cita se stesso. Solo che, al posto di codificare il challenge per lui, C codifica il challenge per A.

Quindi l'attacco deriva dal fatto che il TTP ha tirato fuori  $N_b$ , ma questo usando la chiave non di A ma di C. Questo fatto TTP non lo rileva e nemmeno B ne sarà a conoscenza. Proprio grazie a tutti questi passaggi, C riesce ad ottenere il challenge originale. B a questo punto è fregato, perché B ritiene che il tutto provenga da una determinata sessione.

Nel protocollo descritto prima, l'ambiguità sta nel fatto che, nel messaggio 5, B riceve una nonce codificata, la quale era stata estratta da TTP, ma non è chiaro se TTP l'abbia estratta utilizzando la chiave di uno o dell'altro agente. Quindi l'attacco si verifica proprio perché B, il ricevente, non ha chiaro se  $N_b$  era stato ottenuto per decodifica con la chiave a lungo termine o di A o di C.

## Possibili soluzioni??

Il problema descritto prima si può risolvere. Basterebbe che TTP dicesse a B, nel messaggio 5, il nome dell'agente la cui chiave ha utilizzato. Se questo accade, B non avrà più l'ambiguità e quindi potrà discernere se sia stato l'attaccante o meno. Supponiamo che il messaggio 5 fosse questo:

$$5. \text{ TTP} \rightarrow B : \{A, N''_b\}_{k_b}$$

In questo caso il tentativo di attacco come cambierebbe? Fino al messaggio 4' tutto funziona come sempre. A un certo punto tiriamo fuori dal messaggio 5,  $N''_b$ , usando la chiave di A. Il messaggio 5' sarebbe uguale:

$$5'. \text{TTP} \rightarrow B : \{C, N_b\}_{k_b}$$

da qui tiriamo fuori  $N_b$  utilizzando la chiave di C.

Allora sulla prima sessione, B rivedrebbe indietro  $N''_b$  associato ad A e farebbe abort, perché B aveva associato ad A, nel messaggio 2, la nonce  $N_b$ . Nella seconda sessione, B rivedrebbe indietro  $N_b$  come associato a C e farebbe abort perché B aveva associato a C, nel messaggio 2', la nonce  $N'_b$ .

Quindi farebbe aborto in entrambi i due casi.

Modificando il protocollo, ovvero modificando la forma dei messaggi, riusciamo ad impedire l'ambiguità e quindi l'attacco.

## Possibili soluzioni per il protocollo Needham-Schroder asimmetrico

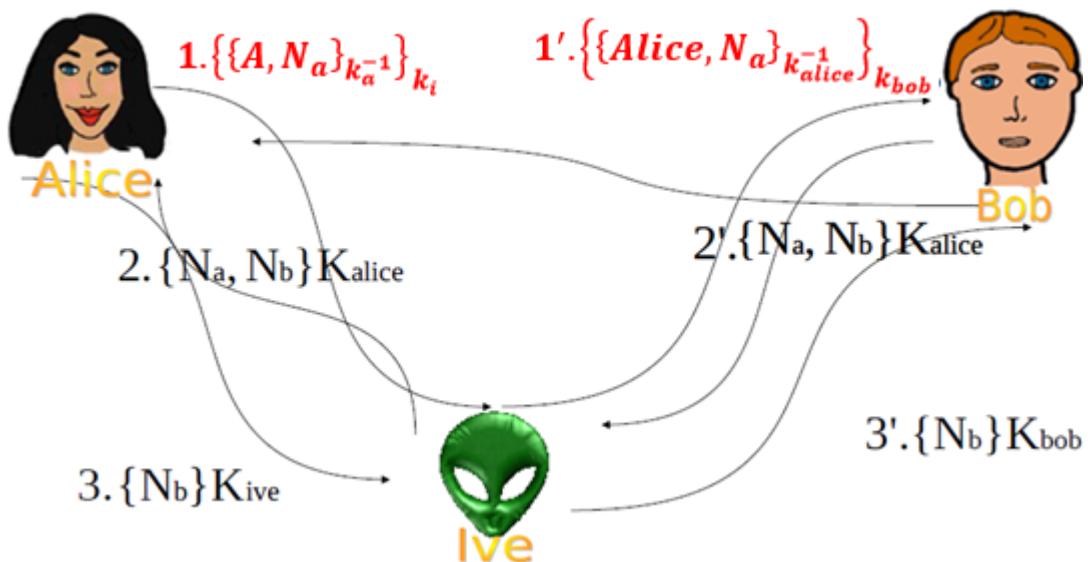
Il protocollo Needham-Schroder è stato descritto in questo modo:



Consideriamo le seguenti possibili modifiche al protocollo. Una o più di queste modifiche potrebbero essere la soluzione all'attacco descritto per questo protocollo:

Numero Modifica	Messaggio modificato	Soluzione?
1° Modifica	1. $Alice \rightarrow Bob : \left\{ \{Alice, N_a\}_{k_{alice}^{-1}} \right\}_{k_{bob}}$	NO
2° Modifica	1. $Alice \rightarrow Bob : \left\{ \{Alice, N_a\}_{k_{alice}^{-1}} \right\}_{k_{bob}}$ 2. $Bob \rightarrow Alice : \left\{ \{N_a, N_b\}_{k_{bob}^{-1}} \right\}_{k_{alice}}$	SI
3° Modifica	2. $Bob \rightarrow Alice : \left\{ \{N_a, N_b\}_{k_{bob}^{-1}} \right\}_{k_{alice}}$	SI
4° Modifica	2. $Bob \rightarrow Alice : \{N_a, N_b, Bob\}_{k_{alice}}$	SI
5° Modifica	1. $Alice \rightarrow Bob : \{Alice, Bob, N_a\}_{k_{bob}}$	NO

Consideriamo la **prima modifica**. Il protocollo modificato sarà questo:



Il primo messaggio avrebbe questa forma:

$$1. Alice \rightarrow IVE : \{A, N_a\}_{k_a^{-1}}_{k_i}$$

ovvero codifichiamo il tutto con la chiave pubblica di IVE.

A questo punto IVE può decodificare con la sua chiave privata la parte più esterna e la parte più interna con la chiave pubblica di Alice. Ma IVE non andrà a decodificare il messaggio più interno perché altrimenti non potrebbe continuare l'attacco, perché, se applicando la chiave pubblica di A, tiriamo fuori  $N_a$ , poi dopo come continua l'attacco? Ovvero, qual è il messaggio che deve costruire IVE per fregare Bob? Secondo il protocollo deve costruire questo messaggio:

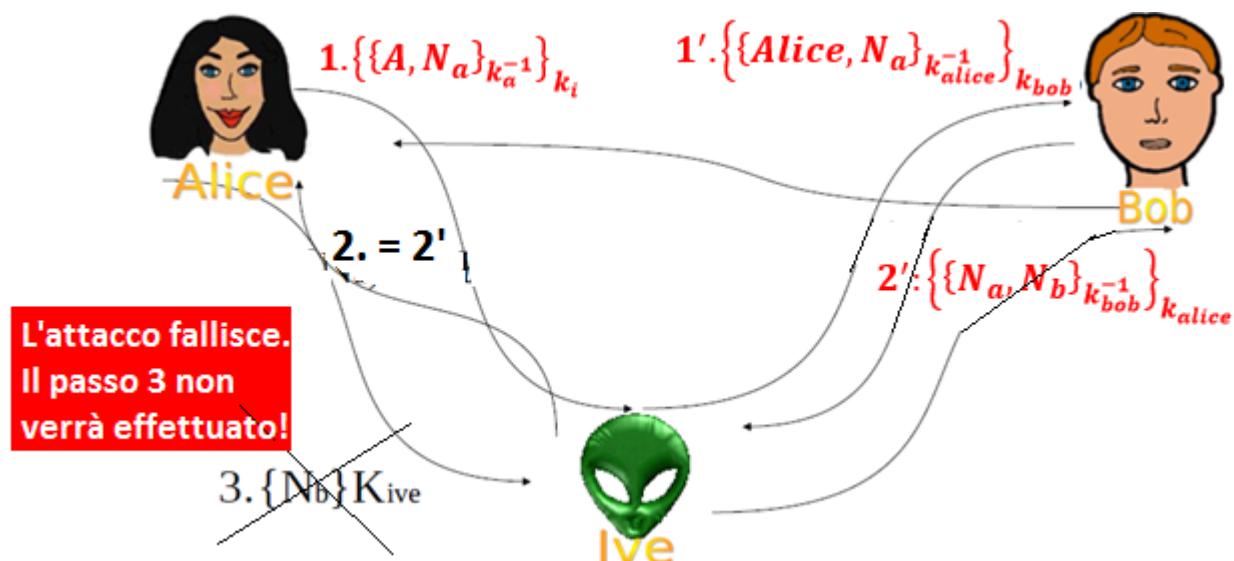
$$(*) Alice \rightarrow Bob : \left\{ \{Alice, N_a\}_{k_{alice}^{-1}} \right\}_{k_{bob}}$$

E l'attaccante si deve attenere a quanto scritto, perché coerentemente con il protocollo, il ricevente pretende di ricevere qualcosa che sia esternamente codificato con la sua chiave pubblica e internamente con la chiave privata del mittente. Quindi

per continuare l'attacco, Ive deve ricostruire il messaggio (\*). A Ive non gli serve tirare fuori la chiave  $N_a$ , perché qualora lo facesse non potrebbe più codificare con la chiave privata di Alice; inoltre non gli serve farlo, perché per fregare il ricevente deve costruire un messaggio della forma (\*). Questo lo può fare. Quindi Ive elimina solamente il livello di encryption più esterno e ricodifica con la chiave pubblica della vittima.

Quindi in sostanza, questa modifica non funziona, perché l'attacco continua; perché Bob alla ricezione di questo messaggio, decodifica con la sua chiave privata, applica la chiave pubblica di A, tira fuori  $N_a$  e costruisce il messaggio 2' e lo scenario di attacco continua come già detto precedentemente.

Consideriamo la **seconda modifica**:



Supponiamo di modificare il messaggio 2' nel seguente modo:

$$2': \text{Bob} \rightarrow \text{Alice} : \{N_a, N_b\}_{k_{bob}^{-1}}_{k_{alice}}$$

Con questo scenario verifichiamo se l'attacco può andare a buon fine. Il messaggio 1 e 1' sono come prima. Il messaggio 2' verrà codificato con la chiave privata del mittente (Bob) e poi con la chiave pubblica del destinatario, ovvero con la chiave pubblica di Alice. Usa la chiave pubblica di Alice perché è specificato nel messaggio 1', quindi Bob replicherà ad Alice. Bob non sa dell'esistenza di Ive, perché altrimenti non ci sarebbe l'attacco. Ive, una volta ricevuto il messaggio 2' non può decodificarlo, e quindi lo manda ad Alice (nel messaggio 2). Alice decodifica con la propria chiave privata e trova il messaggio codificato con la chiave privata di Bob. Alice decodificherà il secondo messaggio con la chiave pubblica di Ive e otterrà un certo  $N'_a$  e  $N'_b$ . Ma Alice non riconosce  $N_a$ , quindi non autentica Ive e l'attacco fallisce. Lo scenario si conclude col passo 2.

Quindi questa è una possibile modifica.

Se consideriamo la **terza modifica**, anche in questa fallisce l'attacco, pur mantenendo i messaggi 1 e 1' come nel protocollo originale, perché è il messaggio 2 che fa gioco, ovvero, appena Alice decodifica con la chiave pubblica del suo destinatario atteso non tirerà fuori  $N_a$  e quindi farà abort.

Questa terza ipotesi di soluzione è soluzione in quanto lo era la seconda. In particolare la terza distilla qual è la modifica che fa veramente gioco.

Anche la **quarta modifica** continua a lavorare sul messaggio 2 (quindi l'attacco fallisce).

La **quinta modifica**, modifica il messaggio 1 in questo modo:

$$1. \text{Alice} \rightarrow \text{Bob} : \{\text{Alice}, \text{Bob}, N_a\}_{k_{bob}}$$

Si è visto che se modifichiamo il messaggio 1, non fa gioco, perché l'attaccante potrebbe mettere al posto di IVE, Bob. Quindi questa modifica non va bene.

Consideriamo dettagliatamente la modifica 3 e 4.

- Modifica 3:

$$2. \text{Bob} \rightarrow \text{Alice} : \left\{ \{N_a, N_b\}_{k_{bob}^{-1}} \right\}_{k_{alice}}$$

- Modifica 4

$$2. \text{Bob} \rightarrow \text{Alice} : \{N_a, N_b, \text{Bob}\}_{k_{alice}}$$

I problemi nascono nel momento in cui l'attaccante impara la nonce  $N_b$ , tanto è vero che, nel primo attacco di segretezza, la nonce  $N_b$  è costruita da Bob, presa e condivisa con A. Nel momento in cui, col messaggio 3, l'attaccante impara la nonce, si ha un attacco alla confidenzialità della nonce. Non a caso poi, l'attaccante riesce a soddisfare Bob impersonando Alice. Dobbiamo impedire che a IVE arrivi il messaggio 3, ovvero dobbiamo impedire che Alice gli mandi quest'ultimo messaggio. Per far ciò, l'ultimo messaggio che arriva ad Alice (prima che mandi il messaggio 3) deve essere un messaggio più saliente, più significativo, più informativo.

**Domanda:** Per quale motivo, modificando il passo 2 del protocollo, non c'è l'attacco?

**Risposta:** È Alice che favorisce l'attaccante dando a quest'ultimo la nonce  $N_b$  nel messaggio 3. Quindi è Alice che ha bisogno di ulteriori informazioni affinché non mandi il messaggio 3. Queste informazioni li deve ricevere dal messaggio 2. Questa è la razzia per cui la terza è una soluzione. Così come su **Woo-Lam** era con il messaggio 5 che a B dovevamo dire qualcosa, qua lo dobbiamo dire ad Alice affinché Alice non estragga la nonce  $N_b$  dal crittotesto.

Quindi con il messaggio 4 accade quanto detto prima, ovvero l'attacco fallisce, perché il messaggio 2 lo costruisce Bob, perché anziché "firmalo" con la propria chiave privata, Bob semplicemente scrive il suo nome, e questo non potrà essere modificato da nessuno perché Bob codifica il messaggio con la chiave pubblica di Alice.

Questi sono correzioni che funzionano! (la 4° modifica è più preferibile della 3° modifica).

## Symmetric Needham-Schroder

1.  $A \rightarrow TTP : A, B, N_a$
2.  $TTP \rightarrow A : \{N_a, B, K_{ab}, \{K_{ab}, A\}_{k_b}\}_{k_a}$
3.  $A \rightarrow B : \{k_{ab}, A\}_{k_b} \leftarrow ticket$
4.  $B \rightarrow A : \{N_b\}_{k_{ab}}$
5.  $A \rightarrow B : \{N_b^{-1}\}_{k_{ab}}$

Questo protocollo usa la crittografia simmetrica ed è stato realizzato dagli stessi autori del protocollo descritto all'inizio del capitolo, ovvero Needham-Schroder. Questo protocollo ha come obiettivo quello di distribuire una chiave usando l'autenticazione. Tutti hanno una chiave a lungo termine condivisa con questo grande fratello **TTP**.

Inizia Alice la sessione con il TTP e specifica che gli interlocutori sono Alice e Bob e aggiunge pure una nonce. Quindi vengono citati gli interlocutori. La nonce viene mandata in chiaro e ricevuta nuovamente nel messaggio 2. L'utilizzo della nonce è per la freshness. Infatti quando Alice riceve il messaggio 2 capirà se è un messaggio vecchio.

Nel messaggio 2 TTP si inventa questa chiave di sessione  $k_{ab}$ . TTP invia ad Alice la chiave  $k_{ab}$ , la nonce di Alice, l'interlocutore atteso di Alice, ovvero Bob, e un **ticket**. Questo messaggio è codificato con la chiave a lungo termine di Alice. Il ticket è il crittotesto codificato con chiave di Bob ed esso contiene la chiave di sessione  $k_{ab}$  ed Alice. Il messaggio 2 è decodificabile solo da Alice. Alice vede la chiave  $K_{ab}$ , che userà con Bob. Questo lo sancisce TTP. Nel messaggio 2 Alice trova il ticket che non può decodificare poiché è stato codificato con la chiave a lungo termine di Bob. Nel messaggio 3 Alice invierà il ticket a Bob. Se fermiamo il protocollo qui vediamo che Alice impara (nel messaggio 2) una nuova chiave di sessione. Questa chiave la scoprirà Bob nel messaggio 3. Ma non ci possiamo fermare qui, perché Bob, alla ricezione del messaggio 3, non ha alcuna evidenza di freshness. Nel messaggio 4, Bob invia ad Alice una nuova nonce,  $N_b$ , e la codifica con la chiave  $K_{ab}$ . Bob non può inviare quest'ultimo messaggio in chiaro, perché nel messaggio 5, Alice decodifica il messaggio ricevuto da Bob e prepara un messaggio da inviare a Bob, ovvero  $N_b^{-1}$  codificato con la chiave  $k_{ab}$ . Il motivo per cui c'è  $N_b^{-1}$  è per distinguere sintatticamente la struttura del 5 dal messaggio 4. Un logico ha detto che, la struttura del messaggio 5 sarebbe stata:

$$\{N_b, N_b\}_{k_{ab}}$$

Questo è un messaggio diverso dal messaggio 4 e ottiene lo scopo di restituire indietro la nonce a Bob. Quindi alla ricezione del messaggio 5, Bob può decodificare e vede la sua nonce. A questo punto Bob capisce che dall'altro lato c'è qualcuno che attivamente può decodificare con la chiave  $k_{ab}$ , in particolare c'è qualcuno posteriormente al tempo in cui Bob ha inventato  $N_b$ . Quindi la chiave  $K_{ab}$  non è più vecchia dell'istante in cui Bob ha inventato la nonce  $N_b$ !

**ATTENZIONE:** consideriamo i messaggi 4/5. Essi dicono:

**<<Caro Bob, c'è qualcuno che usa attivamente  $K_{ab}$  "dopo"  $N_b$ >> (\*)**

Il dopo vuol dire "posteriormente all'istante che hai inventato  $N_b$ ". Invece prima si è detto che la chiave  $K_{ab}$  è recente, ovvero

**<<Caro Bob,  $K_{ab}$  è di "dopo"  $N_b$ >> (\*\*)**

Il significato da dare ai messaggi 4/5 sarebbe l'(\*). Su questa ambiguità sta il nocciolo della questione, ovvero a causa di questo si verifica un attacco di replica. La cosa peculiare è che gli autori hanno preso come significato la (\*\*), ma in base a come è scritto il protocollo l'affermazione vera è l'(\*), perché  $K_{ab}$  è stata inventata prima di  $N_b$ .

Negli anni '90, qualcuno disse questo: supponiamo che l'attaccante voglia replicare l'intera sessione con una ipotesi aggiuntiva, ovvero, siccome questa chiave di sessione è estremamente vecchia, essa può essere violata (per il lungo tempo). Visto che c'è un handshake per la freshness (passi 1,2 e 3 del protocollo), questo è sufficiente a prevenire l'attacco di replica? No! Quindi sostanzialmente questo handshake per la freshness non serve a niente. Allora l'attaccante replica il messaggio 3, (poiché era riuscito a scoprire la chiave di sessione di 2000 anni fa e quindi l'attaccante sta impersonando A) e non appena B fa l'handshake 4 e 5 con una certa sua nonce  $N'_b$  (di ora), l'attacco andrà a buon fine perché verranno eseguiti i punti 4 e 5, e quindi il protocollo si completa. Sostanzialmente con uno scenario di questo tipo, B è convinto di parlare con A, ma in realtà B ha una sessione con l'attaccante, e quest'ultimo è riuscito ad riusare una chiave vecchia come nuova.

Nella sola ipotesi che, essendo la chiave di sessione  $K_{ab}$  di 2000 anni fa, era stata attaccata. Quindi replica + conoscenza di una chiave antica riescono a dare a bere a Bob una chiave di sessione antica come "recente". Questo attacco funziona perché l'handshake non riesce a dire a Bob che la chiave  $K_{ab}$  è "dopo"  $N_b$ , ma riesce a dire a B semplicemente che c'è qualcuno che usa la chiave di sessione dopo  $N_b$ , che è una cosa molto più debole. Quindi malgrado la chiave di sessione sia estremamente vecchia, c'è qualcuno che la usa dopo che B ha inventato  $N_b$ .

Se interpretiamo oggi il protocollo, si capisce che l'handshake finale, non può dire a B che la chiave è fresca, ma semplicemente può dire che c'è qualcuno che la usa di recente. Quindi non ci possiamo stupire se c'è un attacco di questo tipo.

Basta aggiustare il protocollo in modo tale che l'interpretazione giusta dei messaggi sia la prima, perché in questo modo l'attacco di replica non si potrà più fare, perché a B diamo una chiave posteriore al tempo in cui inventa la nonce della sessione corrente.

Il messaggio 1 non è autenticato. Quando TTP riceve il messaggio 1 ovviamente non ha certezza che lo riceve da A. Quindi un attaccante, C, ad esempio, potrebbe scrivere C al posto di A ("Sono C, voglio parlare con B"). C potrebbe lasciare la stessa nonce. Il messaggio 2 sarà codificato esternamente con la chiave pubblica di C. Solo C potrà decodificarlo e quindi A verrà tirato fuori. Questo è un banale attacco di autenticazione, perché l'unica cosa che deve fare C è quello di sostituire A con C. Ma che C abbia agito in questo modo oppure se C cominci il protocollo

legittimamente di suo sarebbe la stessa cosa. Quindi diciamo che è per questo che il messaggio 1 lo possiamo lasciare in chiaro.

## Replay attack

**Def.** Spacciare informazione (*chiavi,...*) obsoleta, magari violata, come recente

Supponiamo che C abbia violato una vecchia chiave di sessione  $K_{ab}$  che B condivise con A

...

3. C → B :  $\{K_{ab}, A\}K_b$  (rispedito identico)
4. B → A :  $\{N_b'\}K_{ab}$  (intercettato)
5. C → B :  $\{N_b' - 1\}K_{ab}$

B autenticherebbe A e quindi accetterebbe di usare  $K_{ab}$ .

# **IPSec**

## Protocolli di sicurezza

IPSec sta al livello IP e rende sicuro il livello IP.

La sicurezza è un'applicazione. Immaginiamo di riuscire a codificare il traffico in uscita che va da noi verso un altro utente e che utilizziamo questo traffico codificato ad esempio facendolo passare verso altri utenti. Poiché il traffico è protetto, se lo facessimo passare verso nodi sicuri, nodi malevoli, non è un problema, poiché non riusciranno ad appropriarsi delle informazioni, ad utilizzarli. Questo è quello che fa, ad esempio, HTTPS. Quindi la sicurezza, in un certo senso, sta a un livello più alto.



Come descritto nei capitoli precedenti, la crittografia è l'arte di trasformare l'informazione scritta in un linguaggio in un altro linguaggio, preservandone la semantica. Ad esempio, la stringa "ciao" la possiamo trasformare in un'altra stringa cambiando l'alfabeto. Un algoritmo di encryption banale sarebbe quello di trasformare quella stringa in un'altra lingua. Se questa non è conosciuta dall'attaccante, esso non tirerà fuori alcuna semantica. Il messaggio sarebbe violabile, nella misura in cui, l'attaccante conosce la semantica del linguaggio del messaggio. L'attaccante che non conosce la semantica del linguaggio del messaggio crittografato, si studia il vocabolario della lingua inglese. Il vocabolario è un'applicazione perché dice come associare una parola a un'altra parola.

Esistono altri strumenti oltre la crittografia:

- **Steganografia:** essa non fa nulla di tutto quello che fa la crittografia. La steganografia nasconde informazioni. Può ottenere un buon livello delle proprietà primarie di sicurezza. Possiamo scrivere un protocollo di sicurezza steganografico ottenendo una delle proprietà di sicurezza, ovvero la segretezza. C'è differenza tra segretezza fatta con la steganografia e con la crittografia. Con la crittografia l'informazione rimane occultata/nascosta all'attaccante. Con la steganografia, il principio è che l'attaccante si potrebbe leggere tutto il libro, quindi in qualche modo potrebbe conoscere quell'informazione ma non la riconosce, non la discerne!

Quindi la steganografia è un buon modo per occultare le informazioni.

- Ad esempio Cambiare i bit meno significativi di un'immagine (digital watermarking)

- **Chaffing & winnowing:** mischiare e separare. Essa è una tecnica un po' ibrida tra la crittografia e la steganografia, perché, non è crittografia in quanto non occulta informazioni, non codifica letteralmente come fa la crittografia. In qualche modo parte dalla nozione di una chiave, di un segreto condiviso. Chaffing & winnowing sembra simile alla steganografia, perché rende l'informazione non distinguibile. Chaffing significa pula (la parte restante, il residuo della trebbiatura dei cereali). Chaffing è il processo di aggiungere questa pula e winnowing è il processo inverso (di selezionare). Quindi immaginiamo di dire un segreto in mezzo tanta pula, cioè come se stessimo nascondendo un ago in un pagliaio. Il principio è quello di dare solo all'interlocutore atteso la chiave, ovvero gli strumenti per trovare l'ago nel pagliaio, cioè gli strumenti per discendere il seme dalla pula. In questo modo l'attaccante avrà seri problemi per trovare questa chiave.

Ecco come si può realizzare.

1. Immaginiamo che i due interlocutori attesi abbiano un segreto. Quest'ultimo è il segreto per trovare l'ago nel pagliaio. Non c'è operazione di encryption. Mittente S e ricevente R concordano una chiave k mediante protocollo Diffie-Hellmann.
2. Se mittente e destinatario hanno questo segreto, il mittente potrà mandare questo segreto al ricevente insieme al MAC (Message authentication control, lo possiamo considerare come un hash crittografico). Questo è un qualcosa simile all'integrità, allo schema della firma digitale. Infatti questo schema è ispirato ai concetti della firma digitale, perché mandando questo MAC, si potrebbe dire che se quello è un digest crittografico, il mittente in un certo senso lo sta firmando. Chiunque nella firma può verificare la firma. Qua no, perché qui la chiave che viene usata non è la chiave privata del mittente, ma è la chiave condivisa tra mittente e ricevente.

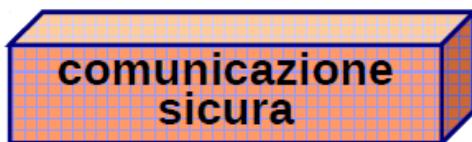
Quindi S spedisce a R coppia **m,MAC(m,k)**.

Se la chiave è condivisa, significa che il ricevente potrà verificare questo MAC come nella firma. Prendo il messaggio, faccio l'hash crittografico e verifico se questo risultato corrisponde alla seconda componente. Questo lo può fare il ricevente. Oltre al ricevente, questa operazione non la può fare nessun altro (a differenza della firma), perché gli altri non avranno la chiave condivisa tra mittente e ricevente.

3. Se il mittente fa solamente questo e l'obiettivo è segretezza, questo obiettivo non lo ottiene, perché il messaggio è in chiaro (perché il mittente manda il messaggio con il digest, ovvero con il MAC). La segretezza si ottiene reiterando questo processo di spedizioni di coppie con coppie random/chaff. Quindi S spedisce a R molte coppie false (chaff) x,y.
4. Il ricevente, riceve tutti questi "messaggi". Per ogni coppia, prende la prima componente, fa l'hashing crittografico per mezzo della chiave condivisa e vede se ottiene la seconda. Questo processo risponderà 'si' solo una

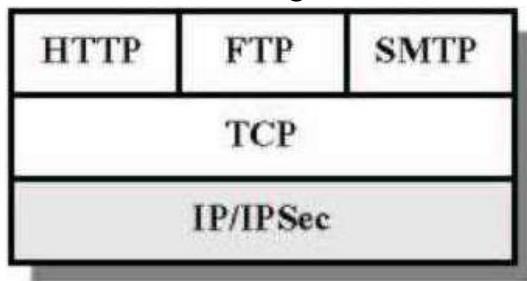
volta, ovvero quando ha beccato il seme. Con la pula risponderà sempre ‘no’, perché è una coppia random, quindi il digest non funzionerà. R seleziona coppia giusta verificandone MAC.

## Sicurezza a livello di rete

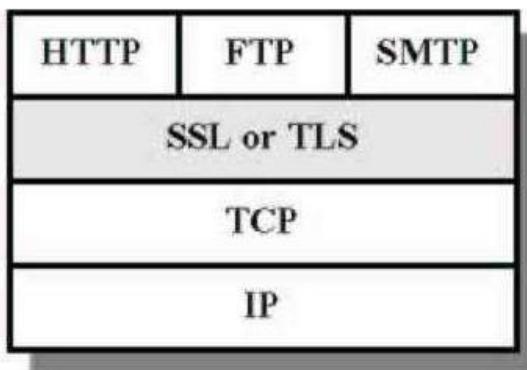


Se avessimo una rete sicura tutte le applicazioni sarebbero sicure. Le proprietà di sicurezza possono essere raggiunte a vari livelli, più in basso andiamo e più la sicurezza diventa trasparente e rigida. Trasparente vuol dire che non ci accorgiamo che c’è il problema, cioè installiamo l’applicazione, lo facciamo girare e il problema di sicurezza viene ignorato, del resto chi è che si accorge che c’è TCP/IP. Rigido vuol dire che non abbiamo margine di customizzazione a livello utente perché è la rete che è fatta in quel modo. Quindi più in basso andiamo e più c’è questo.

C’è chi dice che IPSec non funziona perché è pesante, ovvero appesantisca la comunicazione appunto da facilitare la negazione del servizio.



## Sicurezza a livello di trasporto



SSL sta a livello trasporto o applicazione? C’è chi dice uno e c’è chi dice l’altro. SSL è integrato sul browser al giorno d’oggi, quindi possiamo dire che è a livello applicazione.

## IPSec

IPSec distribuisce una chiave di sessione e questa chiave di sessione codifica i pacchetti. IPSec comprende una suite protocolli per garantire segretezza, autenticazione e integrità a livello IP:

- **IKE (Internet Key Exchange)**: distribuisce la chiave di sessione.
- **AH**: usa la chiave di sessione per fare l'autenticazione
- **ESP**: usa la chiave di sessione per ottenere la segretezza

Quanto detto fin ora su IPSec, è una descrizione molto semplice. Successivamente viene spiegato questo protocollo più dettagliatamente.

Se questo protocollo non è a livello applicazione, ci possiamo aspettare che AH non sia dato in notazione  $A \rightarrow B$ , ma sia uno spaccato di un pacchetto. Mentre invece IKE, che è un protocollo di distribuzione di una chiave di sessione, sarebbe fatto in  $A \rightarrow B$  : "*messaggio*", perché deve distribuire questi messaggi in modo tale che la chiave venga condivisa e IKE andrà su TCP/IP.

Cioè questo protocollo fa questo:

- Prendo una chiave insicura.
- Lancio IKE e distribuisco la chiave di sessione
- A questo punto è possibile fare AH e ESP perché la chiave di sessione distribuita in maniera sicura c'è.

Questa introduzione è stata un modo per parlare di IPSec in maniera *operazionale*.

## Security association

Supponiamo che abbiamo una chiave di sessione condivisa. In questo caso si parla di **Security association (SA)**.

Una Security association è una relazione unidirezionale per una fase di trasporto tra mittente e destinatario. Servono due SA su una comunicazione bidirezionale (ne potevamo fare una per tutte e due, però IPSec è codificato così).

Questa SA indica, sostanzialmente, come si deve trattare questo traffico tra mittente e destinatario (la possiamo considerare una policy in questa misura). In particolare dirà se usare AH o ESP.

Una SA ha almeno i seguenti tre campi:

- Indirizzo IP di destinazione
- Indica se usare AH o ESP
- In particolare ha **SPI (Security Parameter Index)**: un indice che serve ad indicizzare qualcos'altro. Questo dirà in anticipo, a tutti (a tutti coloro che dovrà attraversare per giungere a destinazione), come trattare il pacchetto.

Quindi SPI serve ad indicizzare. L'indice SPI viene utilizzato per accedere al **SAD**.

## Security association database (SAD)

L'indice SPI viene utilizzato per accedere al **SAD** (database delle associazioni di sicurezza), le quali sono attive in quel nodo. Quindi in ogni nodo è attivo un SAD, ovvero, ogni nodo che usa IPSec ha un SAD. Cioè, uso IPSec secondo queste associazioni di sicurezza. L'associazione di sicurezza che devo usare per trattare quel pacchetto che ricevo è dettato da SPI, che viene con la security association. Ogni entry del SAD contiene tutti gli elementi di una SA e in più:

- **AH info:** info aggiuntive sull'algoritmo di autenticazione
- **ESP info:** info aggiuntive sull'algoritmo di codifica
- **Lifetime:** durata della SA.

## Authentication Header (AH)

AH è un pacchetto. Mittente e ricevente hanno una chiave di sessione IKE per poter calcolare MAC (Message Authentication Protocol, in questo contesto lo possiamo considerare come sinonimo di hash, di checksum crittografico, ecc... ). AH autentica l'intero pacchetto esclusi i campi variabili dell'header IP, i quali rimangono modificabili dai nodi intermedi (type of service, flags, fragment offset, time to live, ...), dai nodi attraversati dal pacchetto.

AH serve a fare sia autenticazione, previene sia attacchi di replica e sia alterazione dell'IP.

### Ricordi di "Reti di Calcolatori":

Quando un pacchetto viene mandato esso viene replicato.

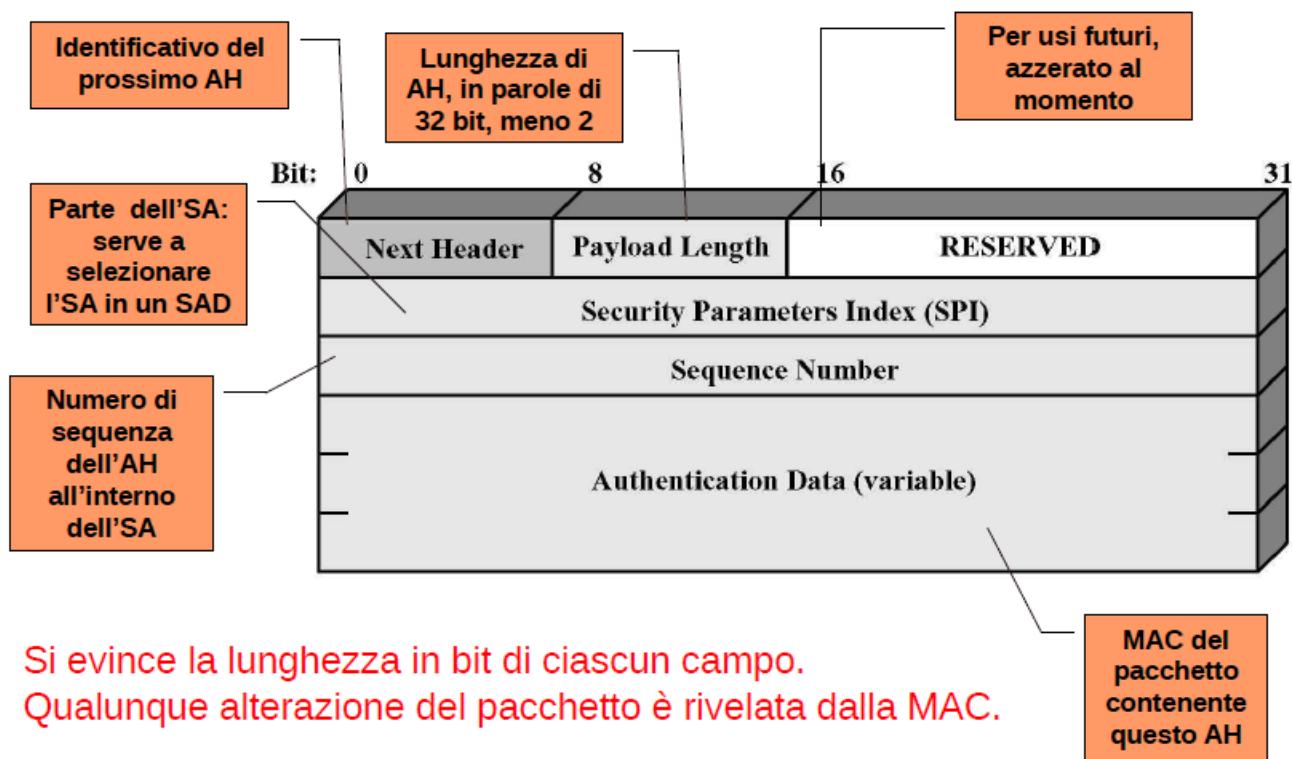
Però prima si è detto che la replica è un attacco. In realtà c'è differenza tra le due affermazioni. Al livello TCP/IP, la replica, si fa per garantire la comunicazione, ovvero garantire che essa vada a buon fine. Qui stiamo considerando il livello applicazione. Un applicazione tende a mandare un intero pezzo di dato, ovvero un messaggio, il quale sarà pacchettizzato da TCP/IP. In sostanza, la differenza sta nell'interpretazione che il mittente dà alla replica. In un caso, il messaggio, che può imbrogliarlo nell'accattare una certa chiave. Nell'altro caso, a livello architetturale più basso, un pacchetto, che è utile a realizzare la connessione.

Quindi in un certo senso la prevenzione dell'attacco di replica si spiega come scritto precedentemente.

Si è spiegato, nei capitoli precedenti, che autenticazione significa riconoscere/verificare qualcuno come dichiara di essere. Per autenticare il pacchetto c'è un campo che ci indica appunto questo. Questo strumento impedirà di alterare in maniera indebita l'IP sorgente. Quindi dire IP spoofing e dire confermerà l'autenticazione sono più o meno la stessa cosa (spoofare significa manometterlo).

## AH – formato

Il protocollo AH è fatto nel seguente modo:



Troviamo:

- C'è un campo riservato
- Next Header: prossimo identificativo AH
- Payload length: lunghezza del payload
- SPI (descritto precedentemente)
- Numero di sequenza
- Dati di autenticazione: essi sono fatti con un MAC.

La parte di autenticazione fa realizzare le proprietà di sicurezza

## IPSec – funzionamento di base

Il funzionamento base di IPSec è il seguente

1. Ogni nodo mittente sceglie una SA usando il proprio SPD (Security Policy Database)
2. L'SPI dell'SA viaggia nel pacchetto AH insieme al Payload originale.
3. Il messaggio 2 arriva a destinazione. Visto tale AH, ogni nodo attraversato ne preleva l'SPI per selezionare dal proprio SAD l'SA relativa al pacchetto.

ESP funziona allo stesso modo. Le proprietà di sicurezza, derivanti da AH, derivano più precisamente da quella parte autenticata. Quindi, come si può organizzare quella parte autenticata, questo digest? Se un attaccante cambia l'indirizzo IP, il digest cambierà e chiunque ottenga questo pacchetto, potrà verificare se il pacchetto/se il payload/se l'indirizzo IP è stato cambiato. In tal caso, il pacchetto verrà rifiutato.

Ma ricordiamo che c'è sempre un digest crittografico, ovvero c'è una chiave condivisa (grazie ad IKE); altrimenti questo discorso crollerebbe.

Questo oltre a realizzare autenticazione, realizza anche integrità. Ma in questo schema non dobbiamo confondere questi due concetti (integrità e autenticazione). La confusione verrebbe poiché stiamo considerando uno schema con autenticazione però di fatto si usa un digest. Esso viene usato per controllare l'integrità sulla firma digitale. I due concetti si avvicinano, laddove realizziamo il controllo di integrità sull'identità del mittente. Cioè il mittente è X, è inalterabile (in mezzo uno strumento di integrità); quindi se applico uno strumento di integrità sull'identità del mittente, ho ottenuto autenticazione.

Riformulando quanto detto prima otteniamo autenticazione con l'integrità per questo motivo:

*A e B si fidano l'uno dell'altro. Se A firma un documento e si ha il modo per prevenire che, chiunque non sia A e B alteri questo documento (controllo di integrità), allora il documento arriverà a B autentico, perché chiunque non sia A e B non potranno alterare la firma di A per farla sembrare quella di A. Quindi il controllo di integrità su una specifica affermazione (identità, mittente), realizza, nell'ipotesi che A e B si fidano, un sistema di autenticazione, perché un attaccante non potrà spaccarsi per A, poiché qualora un attaccante provasse ad alterare quel documento, B se ne accorgerebbe e in quest'ultimo caso rifiuterebbe il messaggio. Pertanto B può autenticare A.*

In questo modo, con il digest, si ha un sistema di autenticazione e integrità.

Quindi abbiamo visto un modo di controllo di integrità come controllo di autenticazione. Però, è anche vero che se il mittente volesse usare un IP fasullo e facesse IPSec con IP fasullo, il destinatario non se ne accorge. Se il mittente vuole imbrogliare il destinatario cambiandosi l'IP, lo può fare, ma dire questo equivale a dire che l'interlocutore è malevolo, e quindi abbiamo scambiato una chiave di sessione per fare un protocollo di sicurezza contro tutto il resto, l'interlocutore è malevolo e quindi con lui non lo farò mai. Quindi se l'interlocutore vuole cambiarsi l'IP se lo cambia. Ma allora a che serviva il meccanismo di integrità descritto prima? Quello serviva per lo spoofing.

Ricapitolando, quando mittente e destinatario sono buoni e si fidano l'uno dell'altro accade che

1. Creano una chiave di sessione condivisa
2. Mettono il proprio indirizzo reale
3. Lo sigillano con il protocollo AH (come spiegato prima), facendo il digest crittografico per mezzo della chiave di sessione
4. A questo punto, i due buoni che si fidano l'uno dell'altro, sono i due che al resto del mondo non possano rompere.

## Prevenzione dei replay attack

I meccanismi a finestra (visti a reti) non nascono per essere sicuri. Un meccanismo a finestra sicuro, se è sicuro otterrà un obiettivo di sicurezza: **impedire attacchi di replica**.

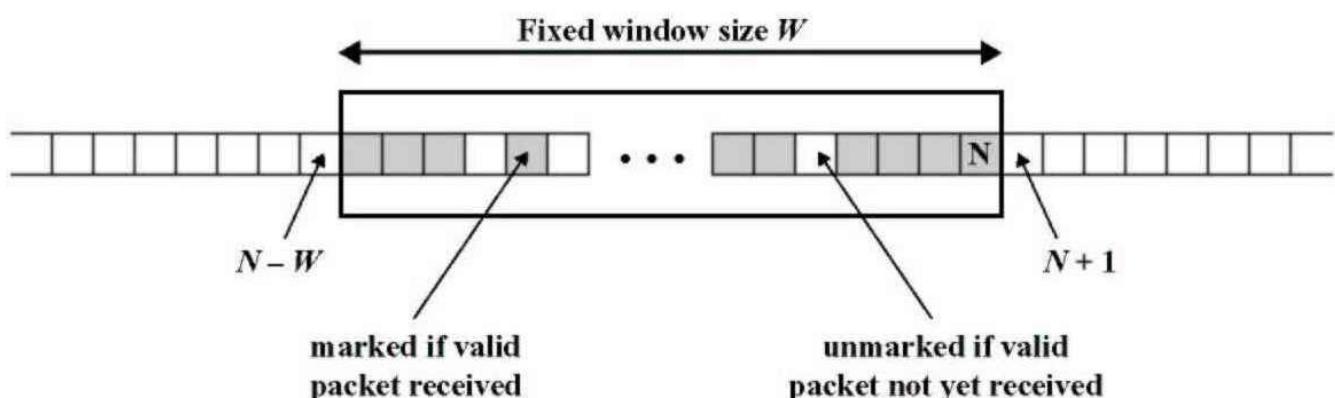
I pacchetti se sono autenticati, possono essere replicati. Per impedire questo, si usa il **Sequence Number dell'AH**.

Il ricevente deve scartare i pacchetti:

- **Vecchi**: classico attacco di replica, ovvero rivendere qualcosa di vecchio come attuale. Questa categoria sono caratterizzati (riconosciuti) perché saranno fuori dalla finestra di accettazione (in particolare saranno anteriori alla finestra).
- **Ripetuti**: saranno già segnati dentro la finestra.
- **Falsificati**: vuol dire, in generale, pacchetto alterato, ovvero attaccato per integrità. Il controllo dell'integrità fallisce. Questo si realizza controllando il digest crittografico.

Quanto detto, è un meccanismo sicuro? Perché mai dovrebbe essere un meccanismo sicuro?

Notare che il numero di sequenza, per quanto detto prima, non si può cambiare per il controllo di integrità. Se si potesse cambiare, nulla vieterebbe all'attaccante di cambiarlo e mandare il pacchetto a destinazione. *Ancora una volta, il controllo di integrità, garantisce/ottiene le altre proprietà di sicurezza.*



Consideriamo una finestra di dimensione  $W$  (figura,  $N$  è il massimo numero). Sia  $X$  il pacchetto che riceviamo:

- Se  $X < N-W$ , il pacchetto verrà scartato perché è vecchio
- Se  $N-W < X < N$ , il pacchetto verrà preso e segnato
  - Se arriva nuovamente un pacchetto segnato esso verrà scartato.

A poco a poco, si avanza con la finestra temporale.

Ricapitolando, per ogni pacchetto dobbiamo controllare:

- Se sta nella finestra attuale (ovvero i due punti descritti precedentemente,  $X < N-W$ , ecc...);
- Che non sia presente;
- Che sia autenticato (cioè il controllo di integrità funzioni).

Se questi tre controlli danno affermativo, accetto il pacchetto. Se invece si riceve un pacchetto troppo nuovo (quindi con un indice alto/superiore all'indice della finestra), la finestra verrà shiftata in avanti.

Quindi questi meccanismi sono arricchiti con il controllo di integrità. Verranno scartati i pacchetti quando essi ha:

- Un indice X inferiore all'indice minimo della finestra
- O già è presente
- O il controllo di integrità fallisce.

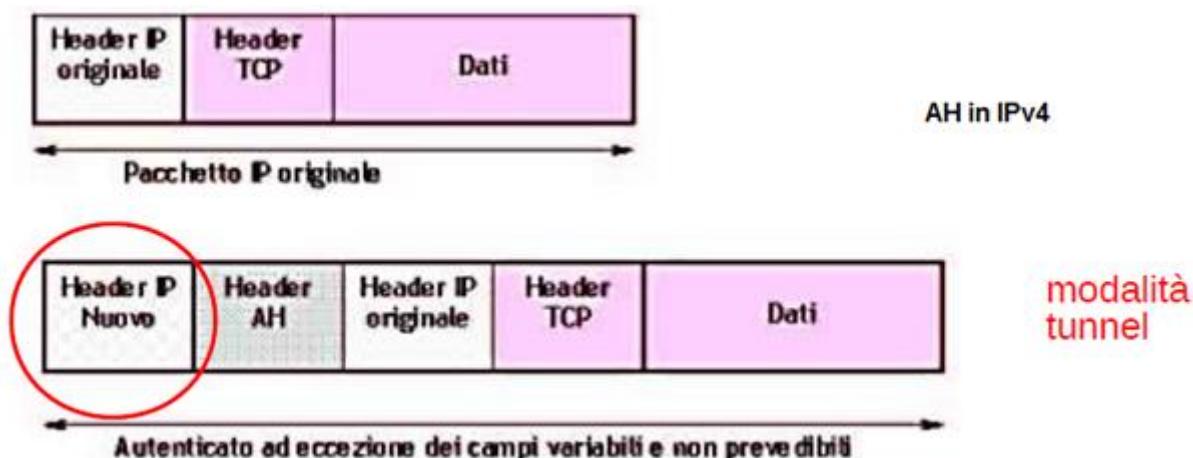
## Protezione mediante AH: trasporto versus tunnel

	Modalità trasporto	Modalità tunnel
Cosa AH protegge mediante il MAC	Dati del pacchetto e campi non variabili dell'header IP (no IP spoofing). In IPv6 anche estensioni dell'header	Intero pacchetto IP (dati + header IP) e campi non variabili dell'IP esterno (no IP spoofing). In IPv6 anche estensioni dell'header

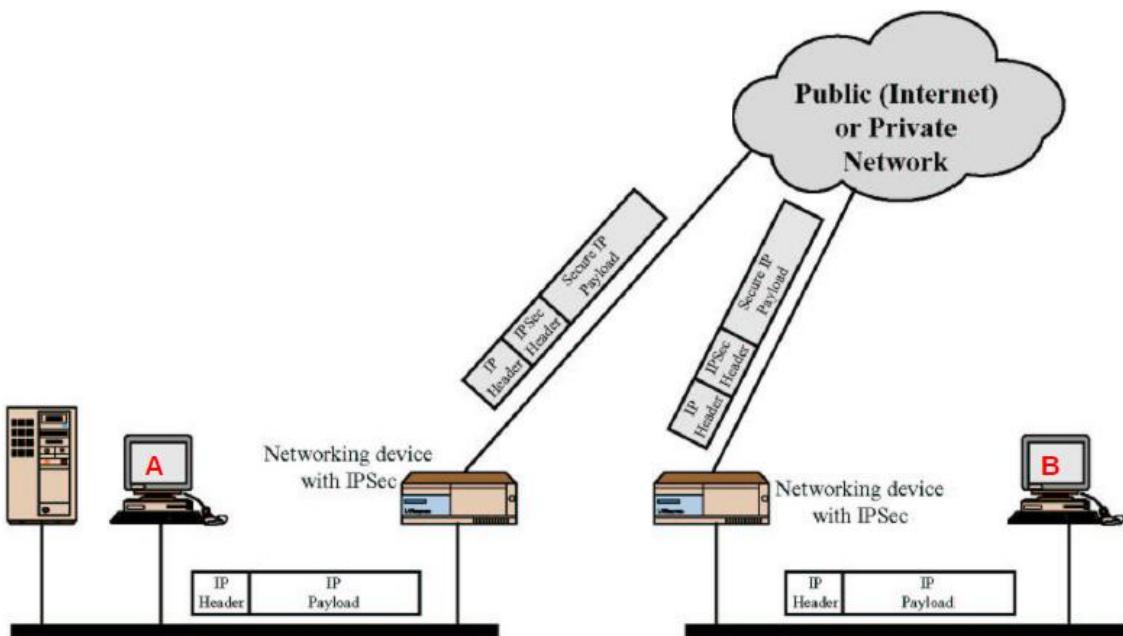
Il  $P_{load}$  per l'header AH (si intende ciò che viene dato in pasto al MAC), è un po' diverso nel caso trasporto e nel caso tunnel:

- Nel caso trasporto abbiamo i dati del pacchetto e campi non variabili dell'header IP, ovvero aggiungiamo l'header AH e manteniamo l'header IP originale.
- Nel caso tunnel abbiamo l'intero pacchetto IP (dati + header IP) e campi non variabili dell'IP esterno; ovvero l'header AH viene seguito da un IP nuovo (ricorda molto TOR).

Nella **modalità tunnel** viene aggiunto un nuovo header IP



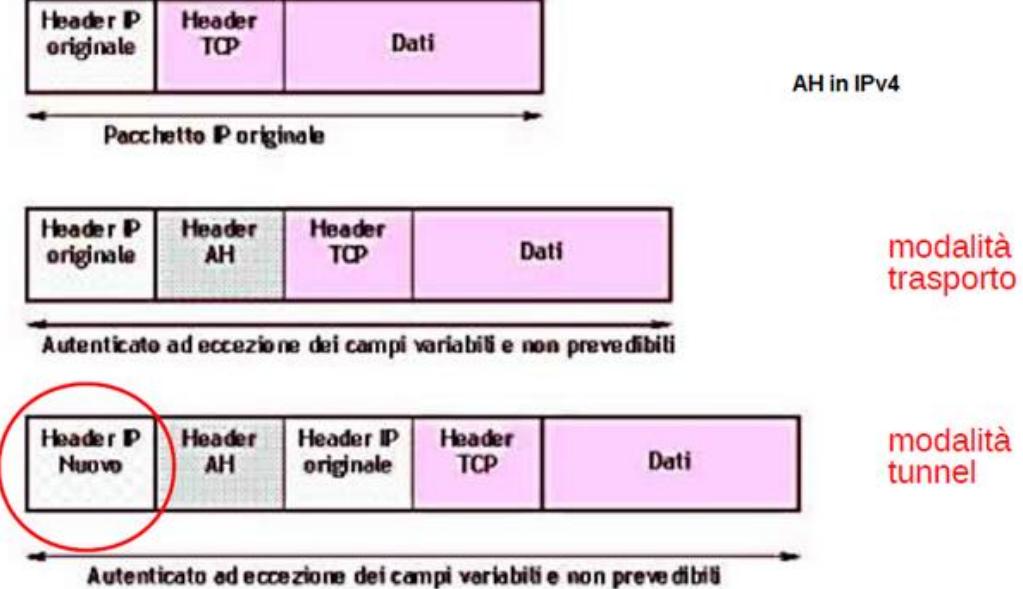
Immaginiamo di avere due macchine, una macchina A sulla rete Catanese e una macchina B sulla rete di Messina. Supponiamo che queste due macchine vogliono comunicare tra di loro.



Entrambe le macchine di IPSec non ne sa niente. I due gateway sono IPSec. La macchina A passa il pacchetto al suo gateway. Quest'ultimo vede che il destinatario è la macchina B di Messina e decide di renderlo sicuro. Ora il mittente è lui (il gateway). Il gateway crea una SA (la cerca nel proprio SPD) e stavolta la incapsula mettendo l'IP del gateway di Messina. Il pacchetto, quindi, arriverà al gateway di Messina. Quest'ultimo prende i pacchetti, fa il controllo di integrità, ecc.. Il gateway, toglie l'header IP esterno (ovvero l'header IP nuovo) e trova l'IP della macchina destinataria. A questo punto il pacchetto viene inoltrato a B.

Questo è l'utilizzo di AH in modalità tunnel.

Cosa si deve fare per ottenere la segretezza? Anziché usare un digest si fa l'encription.

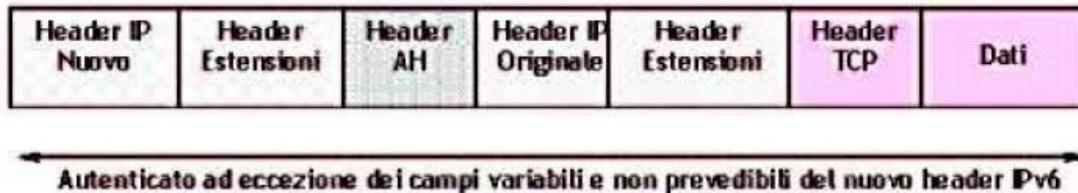




AH in IPv6



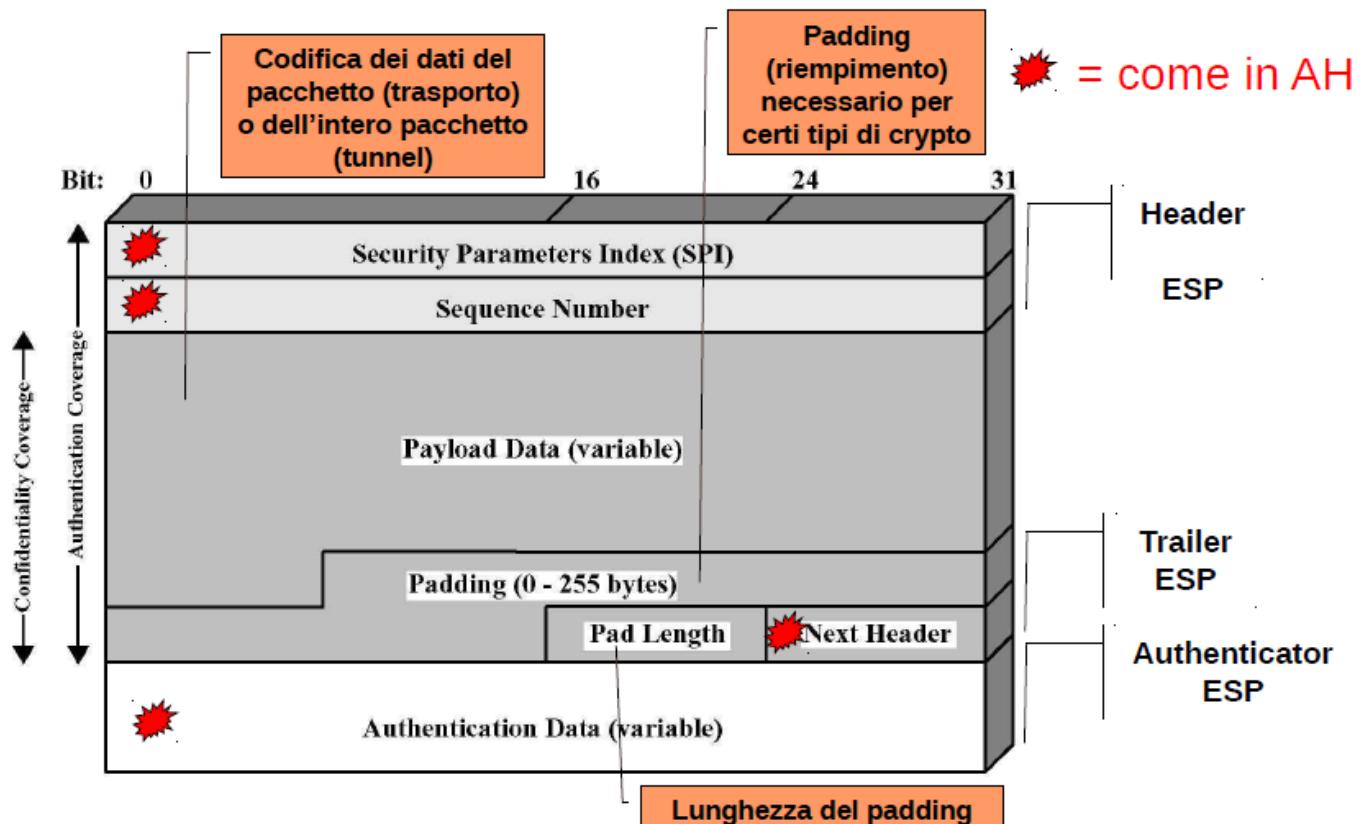
modalità trasporto



modalità tunnel

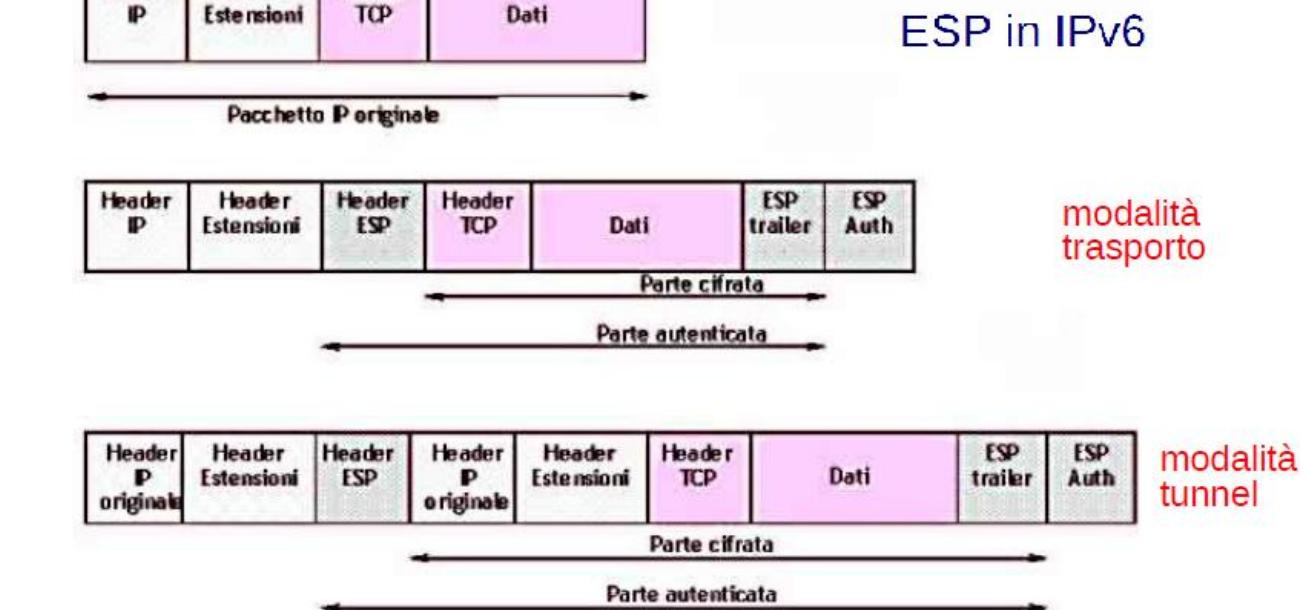
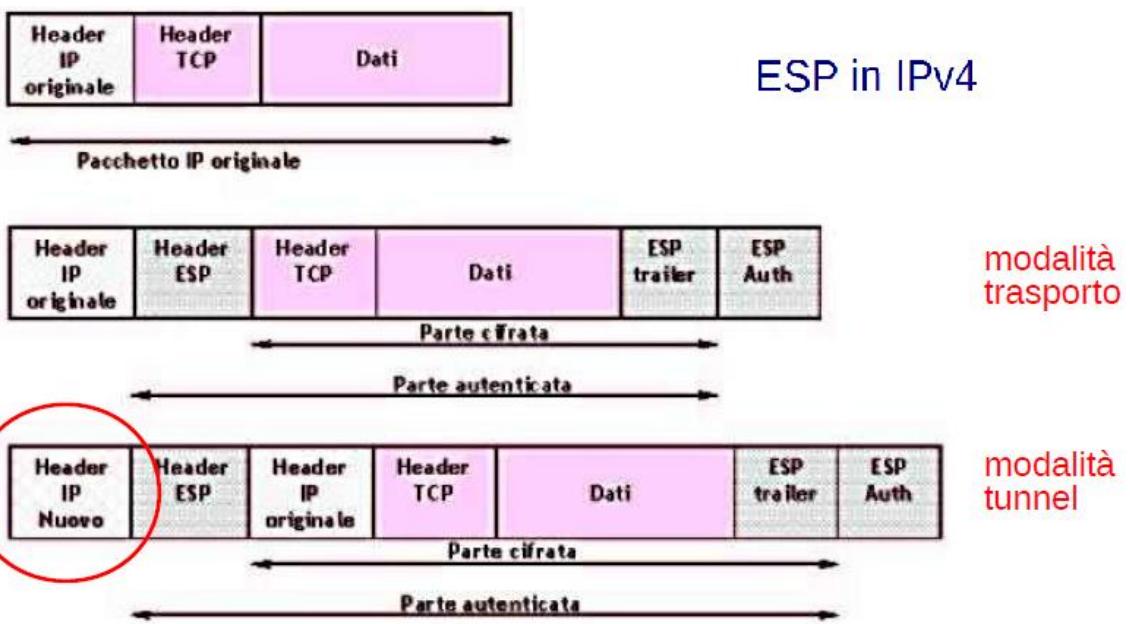
## Encapsulating Security Payload (ESP)

Il pacchetto ESP ha il seguente formato:

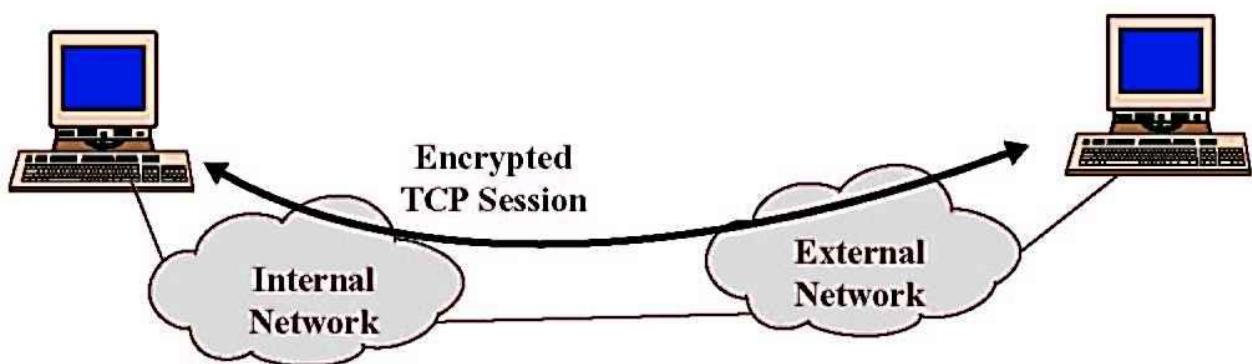


Il Payload del frammento ESP è un crittotesto. Opzionalmente c'è una parte di autenticazione. Da qui possiamo dedurre che, se si mette l'autenticazione si può usare solo ESP, perché ha sia la parte codificata che la parte di autenticazione.

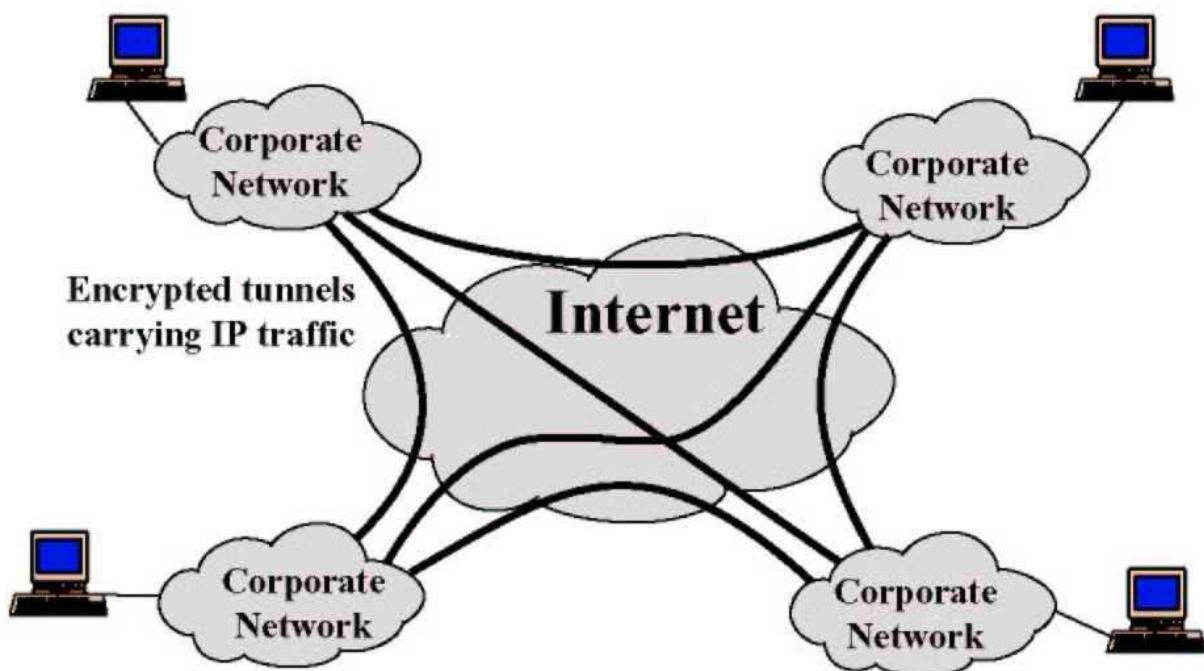
Il discorso **trasporto versus tunnel** è uguale.



## ESP in modalità trasporto



## ESP in modalità tunnel



VPN fra i router/firewall (intermediari) delle varie reti aziendali.

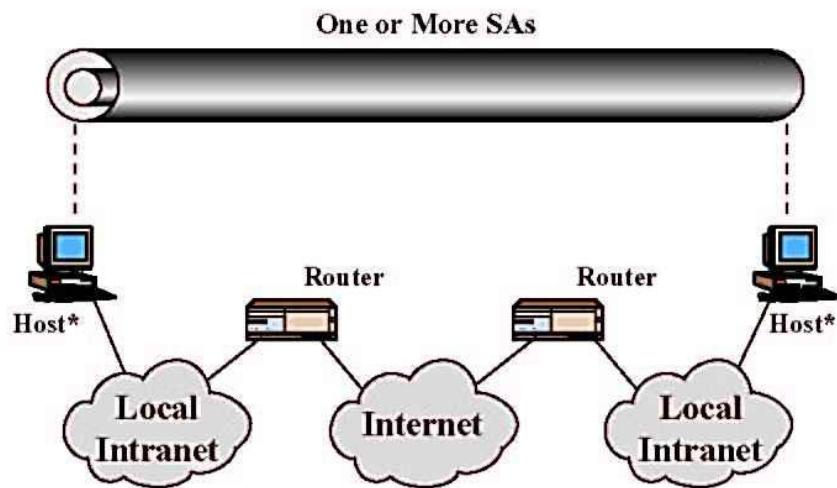
## ESP in tunnel - esempio

1. Nodo A su rete NA genera pacchetto IP classico per nodo B su rete NB
2. Il router/firewall di NA lo incapsula in un pacchetto IPSec modalità tunnel e lo inoltra al router/firewall di NB
3. Questo estrae, **decripta ed eventualmente autentica** il pacchetto IP originale, poi lo inoltra al nodo B su NB

## Combinazioni di SA

Tipicamente una combinazione di SA si fa tra tunnel e trasporto. La specifica dice che ogni nodo IPSec deve supportare questi 4 tipi di combinazioni di SA:

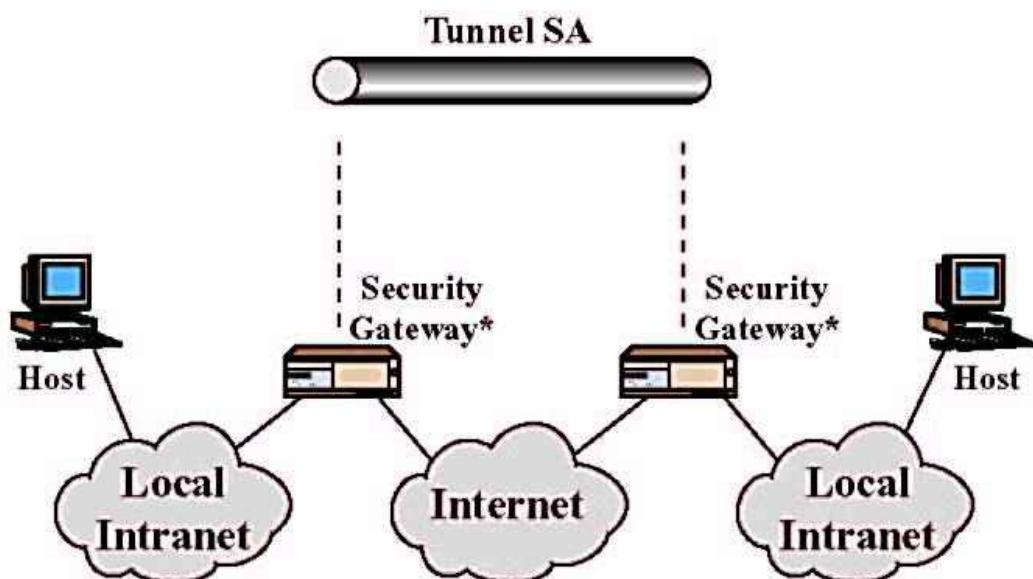
- **Tipo 1: sicurezza punto – punto**



In questa circostanza possiamo avere uno dei seguenti casi:

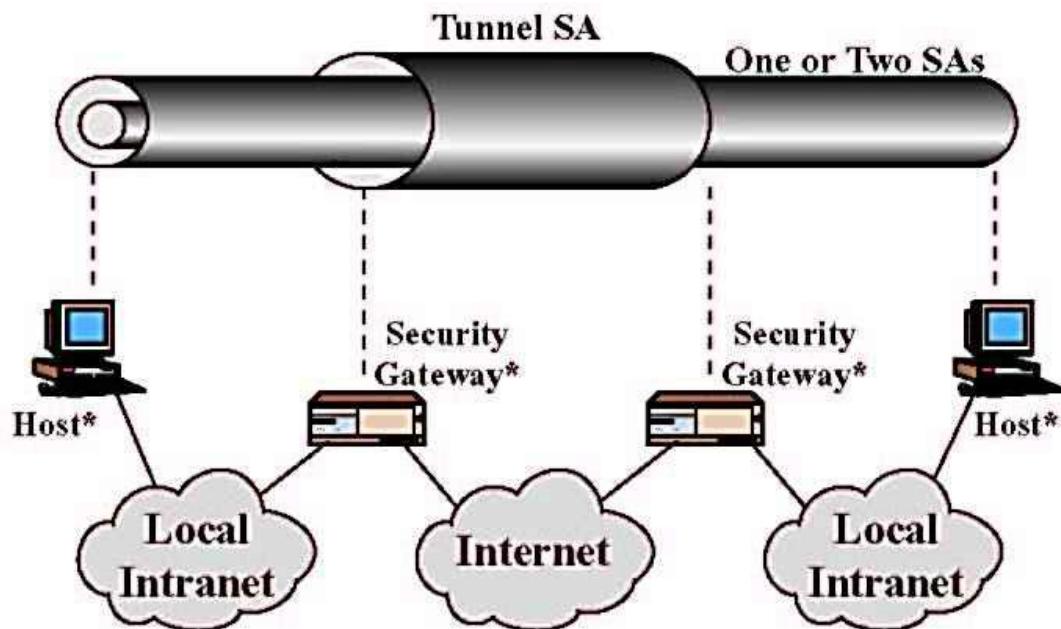
- a) AH in trasporto
- b) ESP in trasporto
- c) Entrambi in traporto
- d) ESP con la parte di autenticazione abilitata

- **Tipo 2: sicurezza fra intermediari**



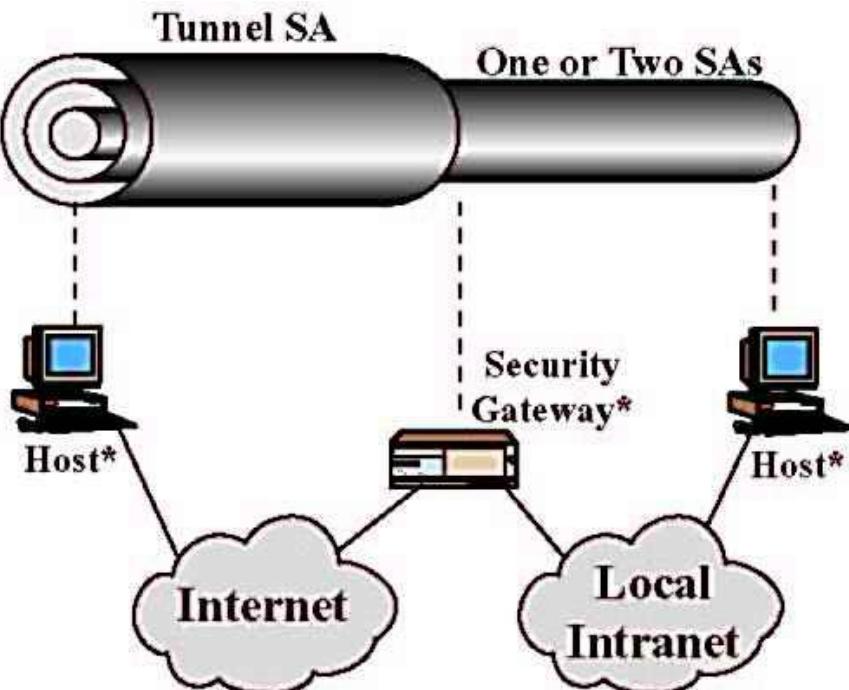
- **Tipo 3: tipo1 & tipo 2**

Questo tipo indica che, non solo si ha il tunnel fra i due gateway, ma si fa pure sicurezza punto a punto fra i due gateway.



- **Tipo 4: tipo 1 & sicurezza punto-intermed.:**

Il tipo 4 indica lo spostamento del tunnel per proteggere dalle minacce interne. Infatti c'è il tunnel tra il mittente e il gateway.



## Internet Key Exchange (**IKE**)

IKE è un protocollo di distribuzione di una chiave.

IKE è la somma di:

- **Oakley:** è una variante di Diffie-Hellmann potenziata (livello applicazione), fornisce chiave iniziale
- **ISAKMP** (Internet Security Association and Key Management Protocol, livello trasporto) è ciò che espande il segreto iniziale in un segreto più lungo in termini di bit.

# **INTRUSION DETECTION**

## Intrusion Detection Working Group

Spesso siamo incappati nella differenza, empiricamente, tra virus e intruso. Con intruso si intende un processo alieno; si intende anche un processo che riesce ad acquistare determinati privilegi.

Rilevare un processo malevolo non è immediato. In qualche modo esso deve essere riconosciuto.

Il rilevamento delle intrusioni è cosa importante. In particolare questo si occupa di:

- Cosa deve fare un **IDS (Intrusion Detection System)**
- Come si debbano descrivere le intrusioni
- Come queste intrusioni impattino i protocolli di comunicazione più tipici

## Intrusione versus Malware

Con virus, sostanzialmente, possiamo indicare un babbone che si attacca ai file esistenti. Mentre un intruso non è un virus. C'è una relazione tra di loro: uno può installare l'altro.

**La caratterizzazione tipica di Malware è di infezione su file.**

**La caratterizzazione tipica di intrusione è di processo in esecuzione.**

DoS	Intrusione
<ul style="list-style-type: none"><li>■ Effetti temporanei</li><li>■ Immediatamente pubblico</li><li>■ Blocco delle risorse</li></ul>	<ul style="list-style-type: none"><li>■ Effetti generalm. permanenti</li><li>■ Spesso non reso pubblico</li><li>■ Uso illecito di risorse</li></ul>

Questa tabella riassuntiva parla di negazione del servizio.

Da qui, diciamo che, stiamo posizionando la questione di intrusione in tutto ciò che conosciamo, rispetto al Malware e rispetto alla negazione del servizio. La differenza con la negazione del servizio è che quest'ultimo è qualcosa di rilevabile, qualcosa che tipicamente abbia un effetto temporaneo, qualcosa che blocca le risorse. In generale, un'intrusione, è qualcosa di più subdolo, che può stare lì per più tempo perché è appunto un processo e che può abusare delle risorse in maniera più subdola, ma anche in maniera più permanente.

Quindi il rischio che su una macchia o su rete esista un'intrusione, un processo intruso che si insinua magari sfruttando una debolezza del browser o del sistema operativo, è alto. Infatti alcuni tipi di intrusioni sul sistema operativo non vengono rilevati dall'antivirus, perché appunto non sono virus ma processi attivi.

## Negazione del servizio (DoS)

**Def.** Attacco che impedisce la normale operatività di un sistema, degradandola o bloccandola del tutto

Possiamo piantare una macchina, puntando a tre obiettivi diversi:

- **cDoS**: consumo di risorse computazionali
- **mDoS**: consumo della memoria
- **bDoS**: consumo della banda di trasmissione

Se facciamo comunicazione di rete e la rete non risponde, siamo piantati. Se facciamo un semplice calcolo che usa molta memoria, il processore sta giù e la memoria ha fame, o viceversa possiamo usare poca memoria e molte risorse di calcolo; in quest'ultimo caso si parla di DoS computazionale.

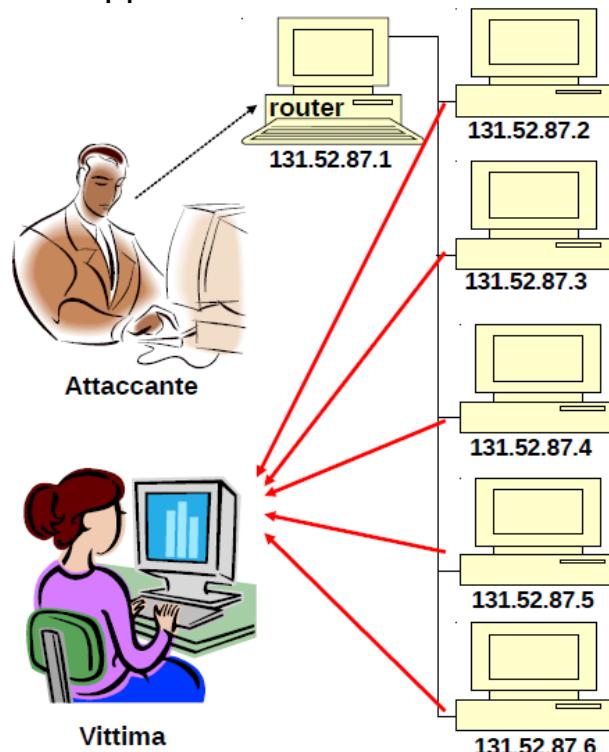
Un DoS può essere anche distribuito laddove sia realizzato da più attaccanti.

### Breve storia

- 1998: primi DDoS
- Febbraio 2000: DDoS contro Yahoo!, Amazon, CNN, eBay. Danni circa \$50M
- 2000/2001: moltissimi DDoS a siti aziendali e governativi USA, e ad ISP europei (a partire da Tiscali UK)
- **21 Ottobre 2002**: attaccati i 13 Root DNS di Internet! Conseguenze minime ma vulnerabilità della rete mondiale appurata

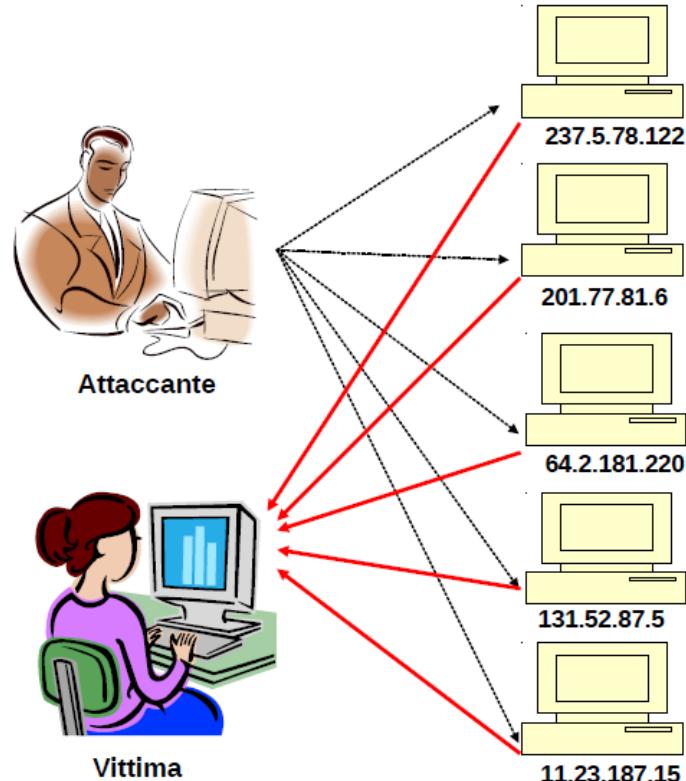
## DoS

- Esiste un router con servizio di broadcast
- L'attaccante invia un PING utilizzando l'indirizzo della vittima
- Tutte le macchine della sottorete rimbalzano il PING alla vittima



# DDoS

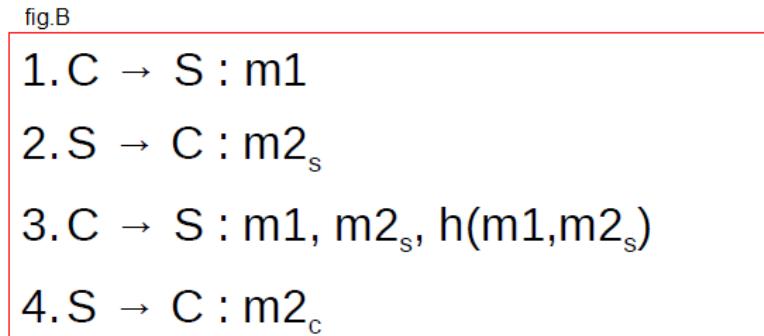
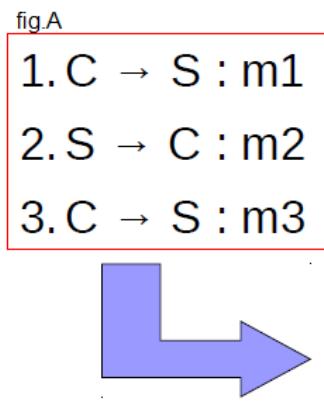
- L'attaccante istruisce un numero di zombie di contattare un preciso IP
- La macchina dietro l'IP viene saturata



## Come reagire?

Un modo per reagire a un attacco di negazione del servizio potrebbe essere l'utilizzo di un firewall, che accetti non tutto, ma accetti periodicamente. Un firewall fa filtri sui porte e quindi possiamo immaginare di scrivere una regola che si metta in ascolto su quella porta solo periodicamente. Questo tempo periodico lo possiamo considerare breve; breve nell'ordine del millisecondo sarebbe già un buon periodo.

La strategia di **cookie transformation**, che in passato è stata implementata in via prototipale, consiste nel trasformare uno schema banale (fig.A) in uno schema meno banale (fig.B):



In figA è presente un protocollo, di protocollo, banalizzato, perché i messaggi in questione sono nascosti (m<sub>1</sub>, m<sub>2</sub>, m<sub>3</sub> generici). Quando è stato spiegato il protocollo Diffie Hellman, si è visto che esso si presta a un attacco di negazione

del servizio; perché se io rispondo a Diffie Hellman, significa che rispondo a fare operazioni di esponenziazione. Quindi se avessi, più o meno in contemporanea, una serie di richieste Diffie Hellman, mi impegnerei su una serie di calcoli si stessi impegnativi. Allora possiamo dire che  $m_1$ ,  $m_2$ ,  $m_3$  generici potrebbero essere impegnativi dal punto di vista computazionale.

Diciamo che, tutta la novità, sta nel discernere/trasformare il messaggio 2 in 2 parti. S sta per semplice e C sta per complesso.

Il messaggio 2, semplice, va dal Server al client come immediata reazione del messaggio 1. Il Server è colui il quale è potenzialmente sotto DoS. Il Server manda al Client un messaggio semplice,  $m_{2s}$ .

Il Client, ovvero colui che starebbe effettuando l'attacco DoS, replica con qualcosa che lo impegna:

$$C \rightarrow S : m_1, m_{2s}, h(m_1, m_{2s})$$

Lo impegna perché deve mandare qualcosa di storico, ovvero il messaggio 1. Inoltre deve fare l'hashing del messaggio 1 e del messaggio 2; che è un modo per costruire quello che in **SSL** si chiama **integrità di sessione** (la sessione, in questo caso, è fatta dal messaggio 1 e dal messaggio 2 semplice). Quindi il messaggio 3 sta richiedendo che il Client possegga il messaggio 1, il messaggio 2 semplice, mandato dal Server, (quindi in sostanza che il Client sia attivo) e fare l'hashing. Solo allorquando il Client costruisca questo messaggio e quindi mostra presenza sull'intera sessione al Server, per mezzo dell'hashing, allora il Server sarà disposto a impegnarsi nel calcolo del messaggio 2 complesso.

Ovviamente il Server può verificare l'hashing e quando questo funzioni, può impegnarsi nella spedizione del messaggio complesso. Ma quanto detto è un'euristica, perché se io fossi un attaccante potrei fare profiling del server. Cioè vediamo cosa manda il Server e vediamo se i messaggi 1 e 2 semplice sono ripetuti oppure sono in funzioni del tempo, se hanno parti fresche. Insomma potrei provare con il profiling a fare una replica, e quindi se riuscissi a fare profiling del Server potrei replicare in maniera "tabellare". Cioè questa tabella la uso per subire il Client; a meno che questi messaggi non abbiano dei marcatori temporali che mi impediscano la replica. Quindi non è che questo schema, astratto per quanto sia, mi risolve il problema principale.

Riassumendo, fig.B, indica che, originariamente il Client cerca di impegnare computazionalmente il Server ed esso si impegnerebbe con Client solo allorquando il Client mostrerà al Server integrità di sessione, ovvero che gli va appresso in tutto il traffico della sessione, e questo è collaborato dall'hashing del messaggio 3.

# Intrusione

**Def.** Ottenere illecitamente privilegi superiori a quelli posseduti lecitamente

Un intrusione è la procedura per mezzo della quale riusciamo ad ottenere privilegi superiori. Acquisire un accesso al sistema oppure ampliare i propri privilegi di accesso sono due esempi di intrusione. Spesso si ridurrà a violare i meccanismi di autenticazione e/o controllo di accesso.

## Tecniche di intrusione

Le tecniche di intrusione li possiamo classificare in questo modo:

- **Violazioni di password:** tecniche standard e non-standard. Le tecniche standard sono gli attacchi dizionario, le parole tipiche, ecc... Le tecniche non-standard sono invece la ricerca di tutto ciò che dipende dal contesto della vittima.
- **Intercettazioni di informazioni sensibili:** ad esempio il monitoraggio del traffico.
- **Uso di combinazioni di SW nocivo:** sullo stile del worm Morris.

## Trattamento delle intrusioni

**Def.**

- Rilevare l'intrusione prima possibile (**intrusion detection**)
- Espellere l'intruso non appena rilevato per minimizzare i danni che può provocare (**intrusion management**)

Il trattamento delle intrusioni si realizza tramite quello che chiamiamo **intrusion detection** e **intrusion management**. Un problema che rimane aperto è la gestione; ovvero una volta rilevato un intruso, come gestiamo questa situazione.

## IDS (Intrusion Detection System)

Per fare rilevamento delle intrusioni dobbiamo fare **auditing**. Un sinonimo di auditing potrebbe essere quello di log di sistema. Auditing significa ispezionare/analizzare i log di sistema.

Prima di fare logging e poi auditing, dobbiamo definire il comportamento di un intruso, cioè dobbiamo registrare nel file di log tutto ciò che possa caratterizzare l'intrusione. Per esempio, possiamo considerare che abbiamo un sistema che registri quante tab nel mio browser apro al giorno. In questo esempio potrebbe accadere che stiamo scaricando software da un sito e contemporaneamente, tramite script, si aprono altre tab. Allora se abbiamo un sistema di rilevamento che traccia quante tab apro al giorno e navigo su un sito per scaricare un software e accade quanto descritto prima, avrei un problema.

In sostanza, c'è bisogno di fare campionamento dell'utilizzo tipico. Il campionamento si registra in un file di log.

L'elemento fondamentale dei log di sistema è un **record**. Un record è un entry in un log. Un record in un entry in un log si dovrebbe fare così: la letteratura dice che tutti i sistemi operativi ha una parte di registrazione di logging. Tramite essi è possibile ispezionare lo storico degli eventi salienti del sistema. Anche questo è violabile con un certo tipo di attacco remoto.

Per i record specifici per il rilevamento abbiamo SW aggiuntivo che raccoglie esclusivamente informazioni specifiche per il rilevamento (Deviazione dal modello statistico). Appesantiscono il sistema, ma danno informazioni mirate.

Tutti i sistemi operativi hanno dei record nativi, in particolare possono avere in aggiunta dei record specifici per rilevare le intrusioni.

Esempio di record specifici per rilevare le intrusioni si chiama **Record di Denning**.

## Record di Denning

L'idea di Denning è quella di acquisire informazioni aggiuntive, così i log saranno più completi. L'auditing avrà più informazioni. In questo modo è possibile stabilire se ci sono delle intrusioni.

I record di Denning sono costituiti dai seguenti campi:

1. Un **soggetto**, che vuole fare una certa azione su un oggetto.
2. **Azioni**: operazione svolta dal soggetto.
3. **Oggetto**: ciò su cui viene svolta l'azione
  - Un soggetto può essere oggetto
4. **Eccezione**: se e quale eccezione sia stata prodotta in risposta all'azione.
5. Uso **risorse**: quali e quanti elementi usati
  - Numero righe stampate o visualizzate
  - Numero di settori letti o scritti
  - Tempo CPU impiegato
  - Unità di I/O utilizzate
6. **Timestamp**: identificatore temporale per il momento di inizio dell'azione

Ecco un esempio:

- Il comando **cp ~giamp/slides.pdf ~barba**  
Genera i 3 record relativi alle 3 azioni:

giamp	execute	/bin/cp	0	CPU = 00002	11058721678
giamp	read	~giamp/slides.pdf	0	SECTORS = 5	11058721679
giamp	write	~barba	-1	SECTORS = 0	11058721680

↑  
anomalia

Questi record sono coerenti con ciò che ha descritto Danning. Nel mio sistema operativo posso andare a vedere se ci sono record di questa forma (o forma simile). I record (log) di sistema, sono una risorsa sensibile! Quindi esse devono essere protette.

Nei nostri sistemi operativi, una volta che inseriamo la password, possiamo fare qualsiasi cosa. Quindi significa che un sistema di autorizzazione nei nostri sistemi operativi, praticamente non c'è (autorizzare, controllare l'accesso sono un po' la stessa cosa).

## Tecniche di rilevamento

Finora abbiamo ottenuto record e sistemi di logging. In qualche modo essi si usano.

- **Tecniche assolute** (indipendenti dal passato): massimo 3 fallimenti di inserimento di password e ti pianta il sistema. Questo perché se si sbaglia più di tre volte, realisticamente, dovrebbe essere un intruso.
- **Tecniche di rilevamento dipendenti dal passato:** si osserva il funzionamento e si assume che il futuro sia come il passato. Questo somiglia molto alla memoria cache.

Queste sono solo euristiche che a volte si verificano e a volte no.

Il rilevamento può essere **statistico** o a **regole**. Snort (NIDS, Network Intrusion Detection System) fa rilevamento a regole.

- Il **rilevamento statistico** crea un modello statistico **utente buono**. Se l'intruso si discosta da questo modello segnalo un attacco altrimenti no.
- Il **rilevamento a regole** crea delle regole per stabilire il comportamento di un intruso.

Questi modelli sembra che si somiglino, perché anche nel modello statistico c'è una meta regola universale, ovvero se ti discosti dal modello oppure no. La differenza in realtà c'è, perché nel rilevamento a regole c'è un grosso insieme di regole, infatti con Snort bisogna aggiornare il database delle regole, come nell'antivirus bisogna aggiornare il relativo database per i virus.

## Rilevamento statistico – modelli

Con il rilevamento statistico possiamo usare dei modelli statistici per caratterizzare le intrusioni:

1. **Media e deviazione standard:** calcolati su un parametro in un arco di tempo, danno l'idea del comportamento medio e della sua variabilità

**2. Modello operativo:** definisce un limite di accettabilità per un parametro e segnala una intrusione quando viene superato. Ad esempio quando diciamo che il limite di creare una password è 3, stiamo usando un approccio statistico basato su questo modello, ovvero l'idea di avere una soglia oltre al quale discernere o meno è il modello statistico che si chiama modello operativo.

**3. Multivariazione.**

**4. Processo di Markov.**

**5. Serie temporale**

**Esempi:** le immagini sotto riportano alcuni esempi di quanto citato prima. Ad ex: fallimenti dell'inserimento delle password. Un modo/modello casuale per riscontrare intrusioni su questa risorsa (parametro) potrebbe essere il modello operazionale che sancisce una soglia. Quantità di output a locazione; quindi ad esempio quanta roba mando sulla porta SSH; media e deviazione standard potrebbe essere usato per rilevare intrusioni su questa risorsa. In particolare, tutte le risorse che hanno un valore assoluto, in questi esempi viene proposto di applicare il modello operativo, perché ad esempio abbiamo 3 tentativi di login e questi sono (non ne possiamo fare di più).

#### Parametro

#### Modello

#### Tipo di intrusione rilevata

Login and Session Activity		
Login frequency by day and time	Mean and standard deviation	Intruders may be likely to log in during off-hours.
Frequency of login at different locations	Mean and standard deviation	Intruders may log in from a location that a particular user rarely or never uses.
Time since last login	Operational	Break-in on a "dead" account.
Elapsed time per session	Mean and standard deviation	Significant deviations might indicate masquerader.
Quantity of output to location	Mean and standard deviation	Excessive amounts of data transmitted to remote locations could signify leakage of sensitive data.
Session resource utilization	Mean and standard deviation	Unusual processor or I/O levels could signal an intruder.
Password failures at login	Operational	Attempted break-in by password guessing.
Failures to login from specified terminals	Operational	Attempted break-in.

#### Parametro

#### Modello

#### Tipo di intrusione rilevata

Command or Program Execution Activity		
Execution frequency	Mean and standard deviation	May detect intruders, who are likely to use different commands, or a successful penetration by a legitimate user, who has gained access to privileged commands.
Program resource utilization	Mean and standard deviation	An abnormal value might suggest injection of a virus or Trojan horse, which performs side-effects that increase I/O or processor utilization.
Execution denials	Operational model	May detect penetration attempt by individual user who seeks higher privileges.

Queste sono le risorse legate all'attività di login.

Queste sono le risorse legate a esecuzione di programmi.

Parametro	Modello	Tipo di intrusione rilevata
<b>File access activity</b>		
Read, write, create, delete frequency	Mean and standard deviation	Abnormalities for read and write access for individual users may signify masquerading or browsing.
Records read, written	Mean and standard deviation	Abnormality could signify an attempt to obtain sensitive data by inference and aggregation.
Failure count for read, write, create, delete	Operational	May detect users who persistently attempt to access unauthorized files.

Potrei studiare queste risorse anche con altri modelli.

Ad esempio, consideriamo il tempo dall'ultimo login. Fisso una soglia e potrei considerare un tipo di intrusione di accesso ad account inutilizzati. Supponiamo di avere un account Facebook che non usiamo da tanto tempo. Il tempo intercorso dall'ultimo login a oggi in media è sempre in crescita. Quindi se Facebook avesse un sistema di rilevamento di questo genere e un giorno decidiamo di accedere a questo account, Facebook con questo tipo di rilevamento statistico direbbe che potenzialmente c'è un attacco.

## Rilevamento a regole

Il rilevamento a regole è basato su un grosso database di regole, fino a  $10^4$ ,  $10^6$ . I rilevamenti a regole sono dei rilevamenti molto rigidi. Danno un output così ben posto, pieno di falsi negativi. Più regole si hanno e più falsi negativi si possono avere. Il vero limite di Snort è proprio questo. Quindi un intrusion detection system potrebbe dossare l'amministratore di sistema.

- Grosso database di regole, fino a  $10^4$  –  $10^6$ 
  - Nessun utente dovrebbe leggere i file contenuti nelle directory personali di altri utenti
  - Nessun utente dovrebbe scrivere su file di altri utenti
  - Gli utenti che si connettono al sistema dopo ore spesso accedono a file utilizzati in precedenza
  - Nessun utente generalmente accede direttamente ai dischi bensì usa servizi di alto livello offerti dal S.O.
  - Nessun utente dovrebbe trovarsi connesso più volte allo stesso sistema
  - Nessun utente esegue copie di eseguibili
- Es: ntop, snort
  - A regole ma basati su firme degli attacchi

## Tecniche di protezione

- 1. Limitare l'automazione:** impedire il più possibile il ripetersi di tentativi
  - Dopo 3 inserimenti di password errata, il sistema resetta la connessione
  - Dopo 3 inserimenti di PIN errato, il telefonino blocca la SIM
- 2. Usare esche:** risorse fintizie che attirano gli attaccanti
  - Magari distogliendoli da risorse più importanti
  - O convincendoli a rimanere di più nel sistema
    - transazioni.grossasocieta.com
    - conticorrenti.banca.com
- 3. Usare trappole:** massiccio monitoraggio appena l'attaccante accede all'esca

## IMS

### (Intrusion Management System)

**Def.** L'intrusione va

1. Contenuta (containment)
2. Rimossa (eradication)
3. Riassorbita (recovery)
4. Punita (punishment)

### Intrusion Containment

- Permettere collegamenti solo su VPN
- Monitorare l'attaccante passivamente
- Limitare le sue attività
  - Diminuirgli i privilegi d'accesso
  - Aumentare la protezione delle risorse
  - E se stacchiamo la spina?!

### Intrusion Eradication

- Chiudere tutte le connessione alla rete
- Terminare i processi dell'attaccante
- Bloccare attacchi futuri
  - Chiudere certe porte
  - Disabilitare specifici indirizzi IP

## Intrusion Recovery

- Ovvia tecnica di recovery
  - Memorizzare periodicamente lo stato del sistema
  - Ripristinare l'ultimo stato (sicuro) memorizzato prima dell'intrusione

## Intrusion Punishment

- Azione legale
  - Difficile rintracciare attraverso la rete
- Tagliare le risorse
  - Notificare l'ISP
- Contrattacco
  - Da studiare e valutare attentamente

# **FIREWALL**

## Misure di sicurezza

Il firewall costituisce una difesa perimetrale a scopo puramente preventivo. Con il firewall vorremmo fare in modo che i problemi non occorrono.

### Usare con cura

La seguente immagine la possiamo considerare come il principio del firewall mal configurato.



Vogliamo che l'accesso del sistema sia effettuabile solo attraverso porte gestite dal firewall. Se con il firewall dimentichiamo o configuriamo male delle porte, potremmo lasciare una sorta di bypass per accedere al sistema. Quanto descritto non è una novità. La letteratura spiega che molti attacchi a sistemi sono fatti per mezzo di porte aperte, per mezzo della mal configurazione di un firewall.

Quello che vorremmo è costruire una “muraglia” a fianco delle porte di accesso, in modo tale che, solo quelle porte di accesso, siano percorribili per violare il sistema e quindi gestite dal firewall.

## Firewall

**Def.** Dispositivo che controlla il flusso di traffico fra reti con diverse impostazioni di sicurezza

Le VLan si prestano bene ad esemplificare una sorta di filtraggio interno. Il firewall non fa altro che filtrare il traffico sulle porte. Quindi le VLan si prestano bene proprio perché separano il traffico (per ragioni di efficienza), ma, in particolare, si prestano bene alla configurazione di un firewall, all'inserimento di un firewall.

## Requisiti di un firewall

1. Tutto il traffico passa attraverso il firewall: quindi l'unico filtro disponibile sia il firewall
2. Ci sia un apposita politica di sicurezza
3. Il firewall si trova su una macchina addennizzata: nella letteratura, tale macchina, si chiama di **bastion host**.

Un firewall, lo possiamo quindi considerare, come colui che applica le policy alle porte. Al di là dell'applicare la policy, il problema sta nei contenuti di quest'ultima. Deve essere una policy che funzioni. Quindi, gran parte del problema nell'avere un firewall che funzioni, è esattamente avere una policy.

## Bastion host

Il termine bastion host è un termine della letteratura. Si tratta di una macchina addennizzata. È buona consuetudine separare i servizi fondamentali per macchina. Quindi non avere tutto su una macchina, ma avere una macchina che faccia idealmente solo un servizio, in modo tale che sia più robusta, che abbia le porte gestite, che abbia una sola porta aperta (Ex. La porta per HTTPS).

## Funzionalità di un firewall

Un firewall protegge le risorse interne, ma, dovrebbe proteggere anche le risorse esterne. Fino a un 10-12 anni fa, l'idea di filtrare il traffico in uscita, faceva ridere tutti. Questo perché, l'idea era quella di proteggerci dall'esterno. In realtà siamo tutti interni a una LAN, ma esterni a tutti gli altri. Quindi tutti potremmo rappresentare delle violazioni potenziali per l'esterno. Una **macchina zombie** è proprio una macchina, nella rete, mal configurata. Essa può essere violata. Ma tale violazione avviene principalmente perché in questo modo questa macchina può essere usata come propulsore per traffico malevolo. Quindi dovrebbe essere gestito anche il traffico in uscita.

Un firewall deve anche:

- Monitorare il traffico
- Filtrare i dati. Questo fino a un certo punto, perché se ne occupa di più un IDS
- Creare VPN
- Fare il NATTING, ovvero mappare indirizzi locali in indirizzi Internet (**NAT – Network Address Translator**)
- Fornire IP dinamici (**DHCP – Dynamic Host Configuration Protocol**)

Questo sembra contraddittorio con quanto detto prima, nel senso che la stessa macchina attiva il firewall, fa DHCP e NATTING (cioè nella stessa macchina

troviamo tutte queste cose). Non è contraddittorio, perché chi meglio del firewall può fare questo! Laddove, per esempio, viene tradotto l'indirizzo della rete in indirizzo privato locale, il firewall permetterà di utilizzare delle regole di filtraggio. Quindi tutte le funzioni descritte prima sono un po' funzioni gemelle, si integrano bene insieme.

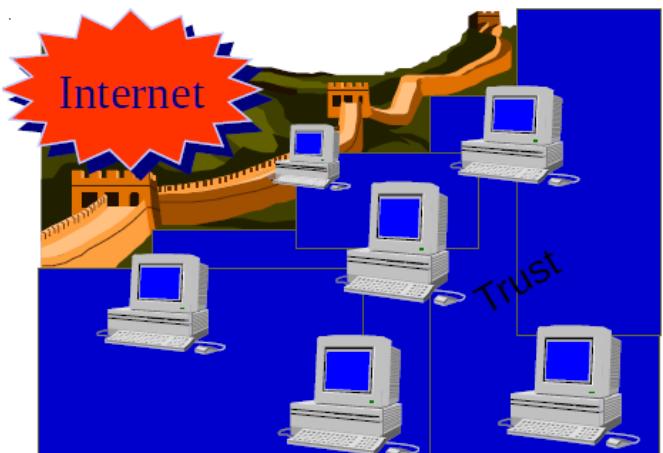
## Politiche di sicurezza adottate

Naturalmente, bisogna considerare le politiche di sicurezza. La politica di sicurezza di default, al giorno d'oggi, è la più limitativa, la più sicura. Quindi con le politiche di default, nego il passaggio (il traffico), chiudo e specifico solo ciò che devo aprire. Questo potrebbe dare dei problemi per la configurazione dei servizi.

- **Default deny:** tutto quello che non è espressamente permesso è vietato
- **Default permit:** tutto quello che non è espressamente vietato è permesso

## La difesa perimetrale \*

Mantenere un'unica porta di accesso ad Internet crea delle relazioni di fiducia fra le macchine che si trovano all'interno della rete



## Limiti di un firewall

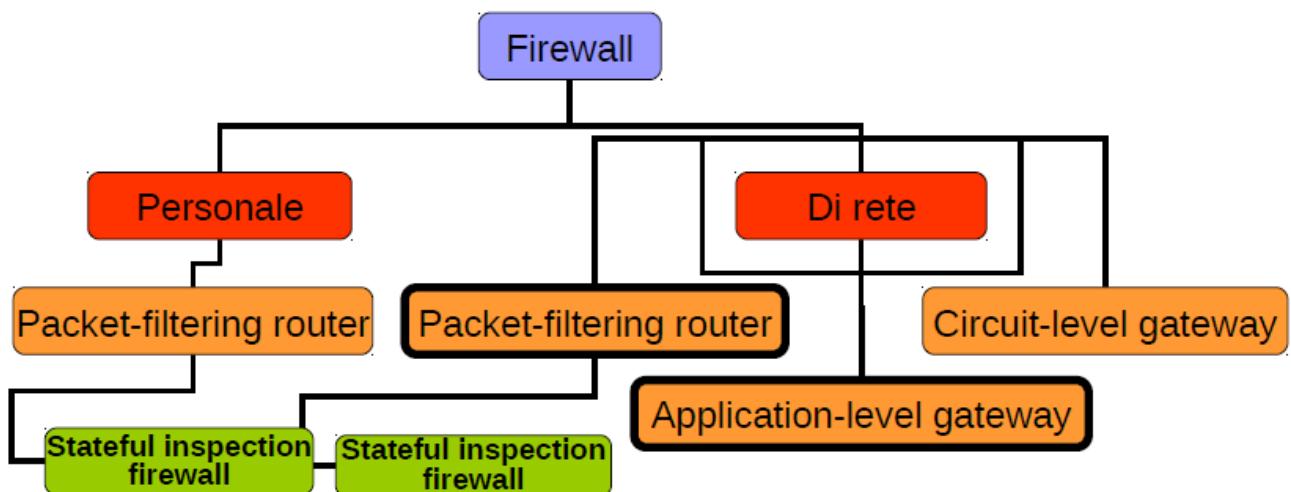
1. Non possono proteggere da attacchi che ricevono il permesso di superare il firewall
  - *Connessioni via modem*
2. Non possono proteggere da minacce interne
  - *Un utente interno potrebbe essere “bad”*
3. Proteggono minimamente dal passaggio di software nocivo. Questo punto vuol dire che un firewall non è un IDS
  - *Setacciare tutto il traffico sarebbe impraticabile*
4. Possono degradare le prestazioni della rete
  - *Filtraggio e monitoraggio*
5. Possono essere difficili da configurare
  - *Difficile compromesso fra libertà e sicurezza*

6. Non possono proteggere da attacchi non ancora documentati ai protocolli di sicurezza

- *Siamo dietro il firewall ma hanno scoperto i dettagli della mia transazione*

## Firewall - tassonomia

La tassonomia cataloga i vari tipi di firewall.



Nella figura si trovano i firewall personali, firewall di rete. Al loro interno si trovano alcune cose comuni ad entrambi le categorie di firewall, ad esempio, il Packet-filtering router. Una specifica del Packet-filtering router è lo Stateful Inspection firewall, che è quella che osserva pure le connessioni.

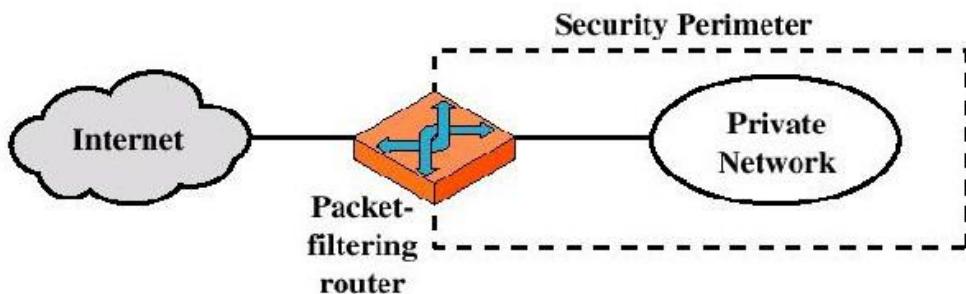
Supponiamo di collegarci alla nostra LAN e supponiamo che sia presente un firewall di rete che monitora il traffico e lo rimbalza, per esempio, per navigare sul web. Il firewall di rete permette al traffico di entrare e di uscire sulle porte dedicate a HTTP e HTTPS. Ma fatto questo, potremmo essere soggetti a un attacco proveniente dall'interno della LAN. In quest'ultimo scenario, il firewall di rete non può intervenire. (Un antivirus non rivelerebbe questo tipo di attacco).

Per risolvere il problema degli attacchi interni, il consiglio sarebbe quello di avere un firewall personale, ovvero avere un firewall sulla singola macchina che gestisca tutte le porte. Un firewall, di fatto, è un gestore delle porte, se un utente dall'interno volesse aprire una finestra, può decidere di sprangare. Le porte si aprono a runtime se l'utente attiva certi servizi, però se c'è un firewall, allora esso farà in modo di non far aprire la finestra, anche se l'utente vuole avviare quel determinato servizio. Quindi il firewall realizza questa difesa centralizzata. Il firewall lo possiamo considerare come una sorta di MAC, nel senso che si presta bene alla definizione di MAC, ovvero c'è un unico amministratore che impone delle regole sui "pass" (nell'esempio di prima, accedere o meno a quel determinato servizio).

## Firewall personale

- Un firewall personale, sostanzialmente, protegge la nostra macchina.
- Tuttora non molto diffuso
- Particolarmente opportuno per utenti mobili (perché se con il nostro dispositivo ci collegiamo alla rete della pizzeria o da qualsiasi altra parte, conviene mantenere il nostro dispositivo sicuro da possibili attacchi interni)
- Utilizzabile insieme a firewall di rete

## Packet-filtering router



Un **Packet-filtering router**, fa filtraggio di pacchetti. Sembra molto simile a SNORT. Ma SNORT lo fa da un'osservazione più specifica, anche sul payload. Un Packet-filtering router applica un insieme di regole che sostanzialmente lavorano a livello di rete, e quindi di fatto guardano gli IP.

Un packet-filtering router guarda i seguenti campi:

1. **Indirizzo IP di origine**
2. **Indirizzo IP di destinazione**
3. **Indirizzi di origine e destinazione a livello trasporto:** numero porta a livello trasporto (TCP o UDP) che definisce i servizi
4. **Protocollo IP:** definisce il protocollo
5. **Interfaccia:** per un router con 3 o più porte, definisce l'interfaccia di provenienza e destinazione

## Packet-filtering router: Esempi di regole

### ▪ Regole Bidirezionali: Esempio 1

- Consideriamo la seguente regola (immagini prese dallo Stallings). Non verrà spiegata la sintassi per scriverle!

action	ourhost	port	theirhost	port	comment
block	*	*	SPIGOT	*	we don't trust these people
allow	OUR-GW	25	*	*	connection to our SMTP port

Le regole specificate sopra indicano:

**1°Riga:** “blocca qualunque porta, qualunque IP, verso SPIGOT”

**2°Riga:** “Concedi al nostro gateway, qualunque sia l’IP, porta 25, il traffico di chiunque dall’esterno”. Quindi di fatto stiamo aprendo la porta SMTP.

La regola di default deny è, in questo caso, implicita. Allora, ricordiamo che, quando si è parlato di policy, si è parlato del problema di come risolvere i conflitti. Si risolvono mettendo le priorità, perché se si trattava di record messi assieme erano logicamente conflittuali. Quindi una regola di default deny dovrebbe essere scritta

- **Regole Bidirezionali: Esempio 2**

- La regola seguente dice che tutte le macchine interne possono collegarsi alla loro porta 25.

action	ourhost	port	theirhost	port	comment
allow	*	*	*	25	connection to their SMTP port

Nessuno mi vieta che sulla porta 25 metto qualcosa costruita ad hoc

- **Regole Unidirezionali: Esempio 1**

action	src	port	dest	port	flags	comment
allow	{our hosts}	*	*	*		our outgoing calls
allow	*	*	*	*	ACK	replies to our calls
allow	*	*	*	>1024		traffic to nonservers

- La prima regola indica che un insieme di IP può, sostanzialmente, uscire verso l’esterno (qualunque porta)
  - La seconda regola dice che se il pacchetto ha questo flag di ACK può entrare
  - La terza regola dice che possiamo uscire sulle porte “alte” (>1024) Questo particolare insieme di regole sta ricalcando un certo tipo di comportamento, il comportamento di **FTP**, poiché viene effettuata una connessione di controllo alle porte basse e trasferimento dati alle porte alte (FTP funziona così). Quindi significa che, l’amministratore di questo firewall deve conoscere come funziona quel preciso servizio e deve cercare di calcarlo a questo livello, che è un livello di espressività piuttosto basso, perché si ha a che fare solo con porte e indirizzi IP. Il problema è che gestire le applicazioni con un linguaggio di espressività così bassa non è facile. Per gestire FTP bisogna scrivere queste 3 regole!

- Consideriamo un insieme realistico di regole:

	Source Address	Source Port	Destination Address	Destination Port	Action	Description
1	Any	Any	192.168.1.0	> 1023	Allow	Rule to allow return TCP Connections to internal subnet
2	192.168.1.1	Any	Any	Any	Deny	Prevent Firewall system itself from directly connecting to anything
3	Any	Any	192.168.1.1	Any	Deny	Prevent External users from directly accessing the Firewall system.
4	192.168.1.0	Any	Any	Any	Allow	Internal Users can access External servers
5	Any	Any	192.168.1.2	SMTP	Allow	Allow External Users to send email in
6	Any	Any	192.168.1.3	HTTP	Allow	Allow External Users to access WWW server
7	Any	Any	Any	Any	Deny	"Catch-All" Rule - Everything not previously allowed is explicitly denied

In questo insieme di regole, l'ultima è la regola di default. In questo caso il sistema è implementato in modo tale da cercare un matching con la prima regola, altrimenti con la seconda, e così via. In questo modo è implementato l'utilizzo prioritario delle regole.

## Packet-filtering router – pro e contro

I packet-filtering in router sono semplici, però hanno un limite di espressività, ovvero quello descritto prima.

Un altro contro è il seguente:

- Mancanza di informazioni di più alto livello nello stack TCP/IP
  - Le applicazioni possono solo essere consentite o meno, le loro funzionalità non possono essere gestite
  - Funzione di monitoraggio inefficiente, raccoglie solo gli elementi visti nelle regole
  - Non supporta autenticazione utente

Il packet-filtering è un linguaggio potente però poco espressivo.

Questo sistema non funziona con IP spoofing. Quindi se riusciamo a fare IP spoofing, non sarà il firewall a fermarci; se mascheriamo il nostro IP in qualche modo, il firewall non noterebbe irregolarità. Utilizzando un sistema, tipo, IPSec, il firewall diventerebbe più affidabile, poiché l'IPSec permette di contravvenire, di limitare l'IP spoofing.

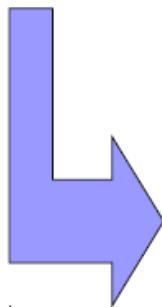
Anche con attacchi di instradamento (routing), il firewall non riscontrerebbe irregolarità.

## Stateful inspection firewall

Una versione un po' più evoluta, un po' più espressiva, rispetto a quanto detto prima, che guarda più ad alto livello di TCP/IP si chiama **Stateful inspection**. Questo tipo di firewall controlla lo stato della connessione, infatti sfrutta informazioni provenienti dal livello di trasporto per gestire meglio le applicazioni. Quindi non bisogna semplicemente dire "apro tutte le porte alte", ma si può essere più precisi.

La tabella del packet-filtering router vista prima, usando lo stateful inspection firewall diventa così:

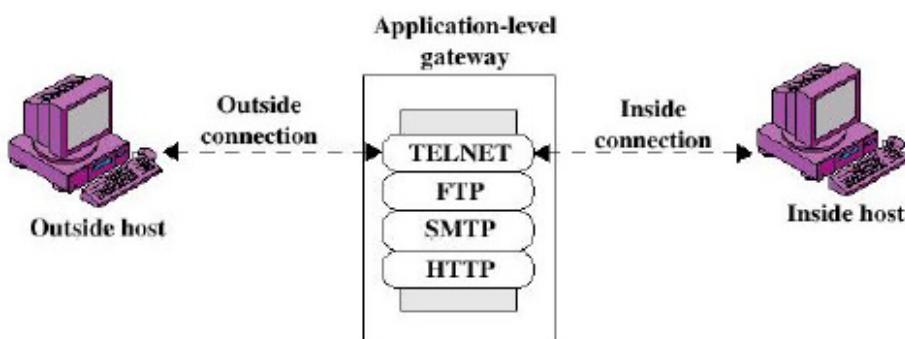
action	src	port	dest	port	flags	comment
allow	{our hosts}	*	*	*		our outgoing calls
allow	*	*	*	*	ACK	replies to our calls
allow	*	*	*	>1024		traffic to nonservers



Source Address	Source Port	Destination Address	Destination Port	Connection State
192.168.1.100	1030	210.9.88.29	80	Established
192.168.1.102	1031	216.32.42.123	80	Established
192.168.1.101	1033	173.66.32.122	25	Established
192.168.1.106	1035	177.231.32.12	79	Established
223.43.21.231	1990	192.168.1.6	80	Established
219.22.123.32	2112	192.168.1.6	80	Established
210.99.212.18	3321	192.168.1.6	80	Established
24.102.32.23	1025	192.168.1.6	80	Established
223.212.212	1046	192.168.1.6	80	Established

Questa tabella più evoluta, ha anche la colonna connessione. Viene controllato lo stato della connessione e in virtù di questo si decide se aprire o meno.

## Application - level gateway

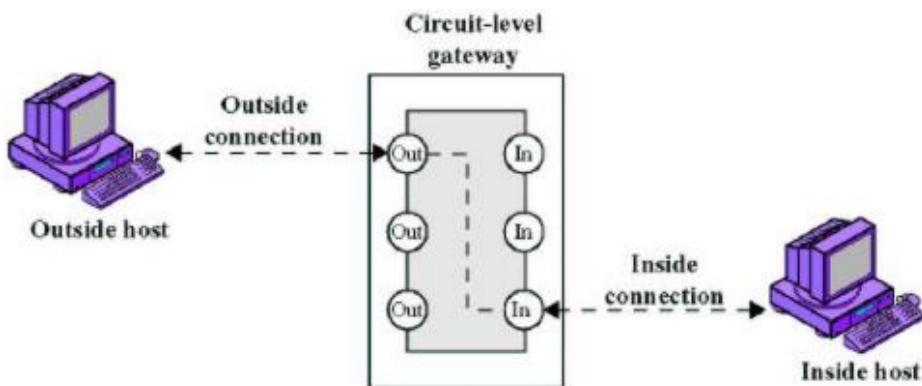


Questo è un firewall a livello applicazione, quindi è un firewall più evoluto, con un software più evoluto, che ci permette di guardare sostanzialmente alle applicazioni.

Ad esempio, possiamo gestire FTP direttamente con questo firewall, anziché dover ricalcare i tratti del loro traffico al livello più basso, come si faceva (come descritto) prima.

Quindi gestendo le applicazioni, si possono gestire anche fonti di autenticazioni utente. Cioè, l'idea sarebbe questa: poiché con l'application – level si possono gestire direttamente le applicazioni, in questo modo si sta implicitamente gestendo l'autenticazione al sistema (al servizio), perché si tratta sempre di applicazioni distribuite che avrebbero bisogno un accesso con autenticazione.

## Circuit-level gateway



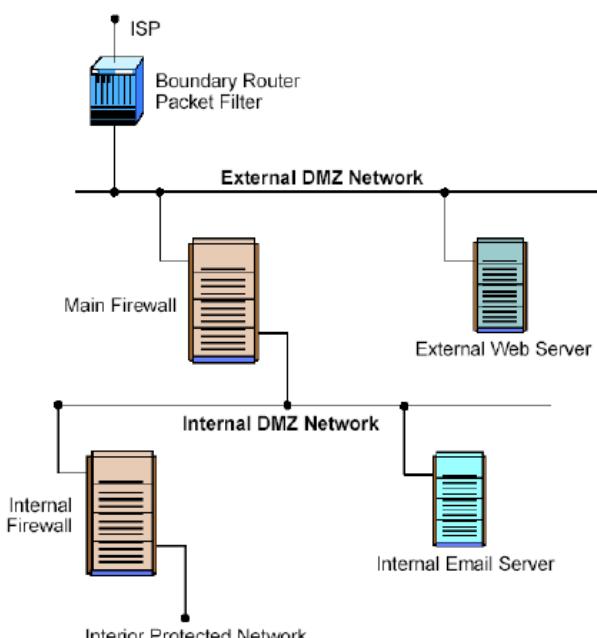
Il Circuit-level gateway funziona con una logica a circuito. È composto da un insieme di porte da un lato e dall'altro e apre due connessioni vere e proprie.

### Nota

Di firewall bisognerebbe utilizzarne un bel po'. Nel DMI ne abbiamo uno solo!

Di seguito vengono descritti dei progetti della rete, che sono complicati per utilizzo comune.

## Applicazione 1: DMZ



**DMZ** significa zona di mezzo, zona demilitarizzata. Nella figura sono presenti due DMZ: una interna e una esterna. Queste zone intermedie stanno fra i firewall di sistema. Questo significa che, viene messo un packet-filtering in router all'esterno, come difesa più primaria, più di punta, e questo packet-filtering in router all'interno ha una DMZ delimitata con un altro firewall. In questo modo i firewall si rimpallano il traffico. All'interno mettiamo, per esempio, un server web in un'unica macchina e quindi il packet-filtering in router instrada semplicemente al server, in modo tale che ad esempio la pagina del dipartimento sia visibile dall'esterno, e tutto il resto del traffico viene mandato a un altro firewall. Quindi la zona demilitarizzata è la zona circoscritta da quel firewall. Non a caso all'interno della zona demilitarizzata, mettiamo delle risorse sensibili, che vale la pena gestire sia come traffico che va verso l'esterno e sia come traffico che va verso l'interno.

## IPTables

IPTables è il firewall standard di UNIX.

Possiamo considerare tre catene: una di ingresso, una di uscita e una di inoltro.

- **INPUT**: utilizzabile per tutti i pacchetti destinati esclusivamente alla macchina firewall e quindi elaborati dai processi locali
- **FORWARD**: Pacchetti destinati ad una delle macchine della rete locale (LAN) o provenienti da essa e dirette all'esterno e non al firewall
- **OUTPUT**: Pacchetti generati dalla macchina firewall diretti all'esterno

# **SOFTWARE NOCIVO (MALWARE)**

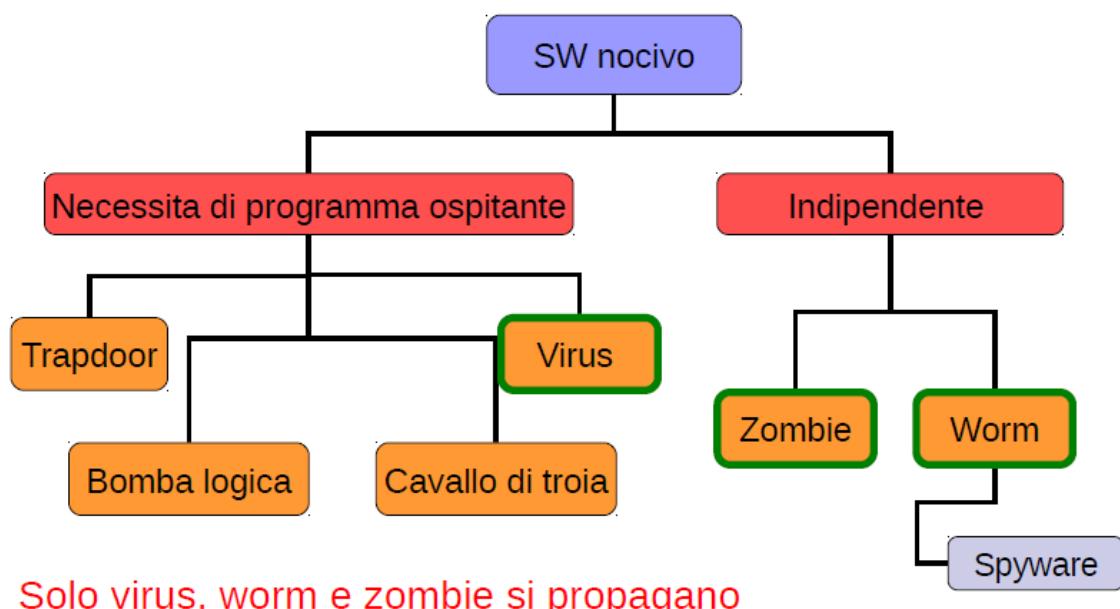
## Bug versus SW nocivo

La differenza fondamentale tra il bug e il malware è la deliberatezza. Il bug è una proprietà inattesa del software. Il malware è deliberatamente scritto per fare danno. Questa sfumatura è alla base della storia conclusiva di un gruppo storici di malware.

Il software nocivo scritto con l'esplicito scopo di violare la sicurezza di un sistema

- Non necessariamente sfrutta dei bug
- Non necessariamente?
  - Ero sicuro non ci fossero bug ma ahimè ho beccato un virus lo stesso

## SW nocivo – tassonomia



Questa tassonomia differenzia il malware che ha bisogno di un ospite (programma ospitante), da quello che non ne ha bisogno.

Ecco i malware che necessitano di un programma che li ospita:

- Trapdoor
- Bomba logica
- Virus
- Trojan

Ecco i malware che non necessitano di un programma che li ospita:

- Zombie
- Worm

Sarà facile confondere virus e worm, ma in realtà, il discriminante tra i due è l'indipendenza dai programmi ospitanti.

## Trapdoor (back door)

**Def.** Punto segreto di accesso ad un programma senza le procedure di sicurezza altrimenti previste

Una **bask door** è un punto di accesso al sistema che sostanzialmente vive dentro il sistema.

Immaginiamo di avere un programma, scritto con un qualsiasi linguaggio di programmazione strutturata, con una serie di annidamenti if-then-else. Se vogliamo provare questo programma abbiamo un problema. Una cosa è avere un programma con pochi annidamenti (vuol dire if-then-else, salti condizionali, cicli), pochi if, e un'altra è averne molti. Se abbiamo molti annidamenti if-then-else, per verificare che tutti i blocchi interni funzionino come noi vogliamo, dobbiamo controllare prima il primo blocco, poi il secondo, e così via.

Allora, storicamente, si sono inventati il **back door**, che, uno degli scopi è stato appunto quello di semplificare il beta-testing. Se abbiamo una back door, scritta dentro quel blocco molto annidato, allora sostanzialmente, abbiamo un salto incondizionato verso quel blocco estremamente annidato. Quindi se vengono inseriti questa serie di caratteri, allora facciamo un salto incondizionato ed entreremo dentro quello specifico annidamento. Chi vuole fare il beta-testing al proprio programma che possiede le caratteristiche descritte prima, allora questa tecnica risulta essere molto comoda. Ma questo è il motivo storico. Va da se, che c'è una sequenza per fare il salto incondizionato, per esempio, ci sarà un controllo: "Se inserisci questa sequenza, allora avverrà il salto". Quindi è un salto incondizionato, ma in realtà è condizionato dalla ben precisa password/ keyword. Prima si è detto che è un salto incondizionato, ma un salto incondizionato previsione non ne ha; è incondizionato perché ha le caratteristiche del salto a sproposito/del salto non strutturato, ma è sotto una condizione, la **condizione di segretezza**. Chi conosce la back door, la può quindi indirizzare (o utilizzare).

Ma abbiamo sempre detto che noi non possiamo immaginare di costruire un sistema sicuro nascondendo le caratteristiche, per il semplice fatto che queste caratteristiche prima o poi verranno esposte.

## Bomba logica

**Def.** Frammento di codice di un programma non nocivo pronto a “esplodere” quando si verificano certe condizioni

Una bomba logica è uno specifico malware che lancia il suo carico (esplode) quando certe condizioni logiche si verificano.

Questo è particolarmente cattivo, perché il file malevolo potremmo installarlo in un sistema che apparentemente funziona, che funziona per un bel po’. Ma il fatto che funzioni per un bel po’, non vuol dire che, ad un certo punto, non possa scattare un elemento di nocività. Per esempio, ci potrebbe essere un controllore sul log di sistema che controlla il numero di login e che decida che a un certo numero di login effettuati con successo, parta un carico cattivo. Questo è un esempio di bomba logica.

Quindi una bomba logica è un qualcosa di occulto, fino a quando non si verificano le condizioni per farla esplodere.

## Cavallo di troia (Trojan)

**Def.** Programma utile o apparentemente utile che in fase di esecuzione compia violazioni di sicurezza

Un trojan è un programma che sembra buono e invece non lo è. (Il nome appunto è quello storico).

Se consideriamo l’esempio di trojan (pag.7), apparentemente sembrava qualcosa di buono, ma invece nascondeva interessi cattivissimi.

## Zombie

**Def.** Programma nocivo che sfrutta una macchina remota già violata per lanciare nuovi attacchi che difficilmente possono essere ricondotti all'autore dello zombie

Uno Zombie è un pezzo di software nocivo che sta al servizio di altro malware, di un controllore centralizzato.

Si è parlato di Zombie in DOS distribuito. Quindi il fatto di avere una macchina violata che può essere usata per attaccare, la rende uno Zombie. Immaginiamo di trovare sulla nostra rete locale una macchina facilmente violabile. Allora un cattivone potrebbe avere un obiettivo peggiore, ovvero, dopo aver studiato la macchina violata, potrebbe usarla per mandare traffico, per esempio, di negazione del servizio. Oppure, l'attaccate dopo aver violato una macchia e aver inserito in essa un processo malevolo, potrebbe capitare che mentre il proprietario la usa, potrebbe (il processo malevolo) a sua volta fare danno altrove!

## Worm

**Def.** Programma nocivo che infetta macchine remote, ciascuna delle quali a loro volta infetta altre macchine remote

Un worm è un programma nocivo che infetta macchine remote, ciascuna delle quali a loro volta infetterà altre macchine remote. Questo non va confuso con la definizione di zombie, perché questo "verme" si replica di volta in volta sulle macchine. Un worm può risiedere sulla macchina in maniera del tutto indipendente. Un processo, un intruso, ha le caratteristiche di un worm. Un virus ha delle sfaccettature diverse, perché esso ha bisogno di un ospite.

Un worm è più dannoso di uno zombie perché si trasmette da macchina a macchina, mentre invece lo zombie diventava lo schiavo, non aveva più l'obiettivo di replicazione. Quindi un worm non fa altro che essere uno zombie con più le caratteristiche di creare un altro zombie.

Il worm più storico è il Morris Worm, dal nome del creatore. Il Morris Worm infettò 6000 machine in ore. Sembra che questo worm non faceva altro che capire dove

andare, andarci e autoriprodursi. (ex. Possiamo considerare un invasore di privacy).

Ciò che scrisse Morris aveva un bug. Alcune versioni del programma non terminavano e quindi calavano le prestazioni della macchina.

Il tribunale si pronunciò sulla deliberatezza di questa azione. Poiché non c'è stata deliberatezza, allora questo problema si è chiamato bug, altrimenti sarebbe stato un malware. A causa del Morris Worm, migliaia di macchine venivano disconnesse dalla rete e molte si piantavano.

Il Morris Worm sfruttò tre bug noti (di quei tempi) per propagarsi:

1. Attacco known-ciphertext su **file di password**. known-ciphertext o conosciuto attacco di testo cifrato è un modello di attacco per crittoanalisi cui l'attaccante si assume di avere accesso solo a una serie di testi cifrati. Significa che se abbiamo un crittosteo, un modo per decodificarlo implicitamente sarebbe provare a codificare un dizionario, fino a quando non arrivo a trovare il crittosteo. Nella crittografia, molti degli algoritmi di encryption hanno randomicità. La randomicità serve proprio a prevenire un attacco known-ciphertext. Al tempo del Morris Worm la randomicità non c'era e quindi c'era questo potenziale attacco.
2. BOF mediante programma finger.
3. Trapdoor nel programma **sendmail**.

Uno di questi tre modi veniva usato per propagarsi. Una volta propagato, l'incarico era quello di copiarcisi dentro. È chiaro che una volta copiato dentro e andava in esecuzione, il worm, non era necessariamente malevolo, non doveva per forza fare danno, poteva semplicemente continuare a fare le sue statistiche. Il problema era capire che cosa veniva fatto con questi tipi di monotoraggi.

L'attacco known-ciphertext si può fare anche su questo: si è parlato dello schema per la registrazione della password anche con sale, però nel momento in cui riusciamo a leggere la password codificata (in versione Hash), possiamo provare l'algoritmo "in avanti" (cioè prendiamo un altro candidato e vedere cosa accade) e il sale dovrebbe aggiungere quella randomicità di cui si parlava prima, ma i bit di sale sono pochi. Quindi si potrebbe provare l'attacco known-ciphertext anche oggi. Non a caso le funzioni Hash sono violabili proprio con questa idea di attacco known-ciphertext.

Il BOF è la cattiva gestione della memoria, la quale causa il trabocco (l'overflow) del buffer.

## Virus

**Def.** Programma nocivo che viola altri programmi non nocivi, sfruttandoli per propagarsi

Un virus è un programma nocivo che fa danno, che sfrutta dei programmi ospiti. Polimorfismo, criptazione, sono tutte tecniche che i virus potrebbero usare per nascondersi.

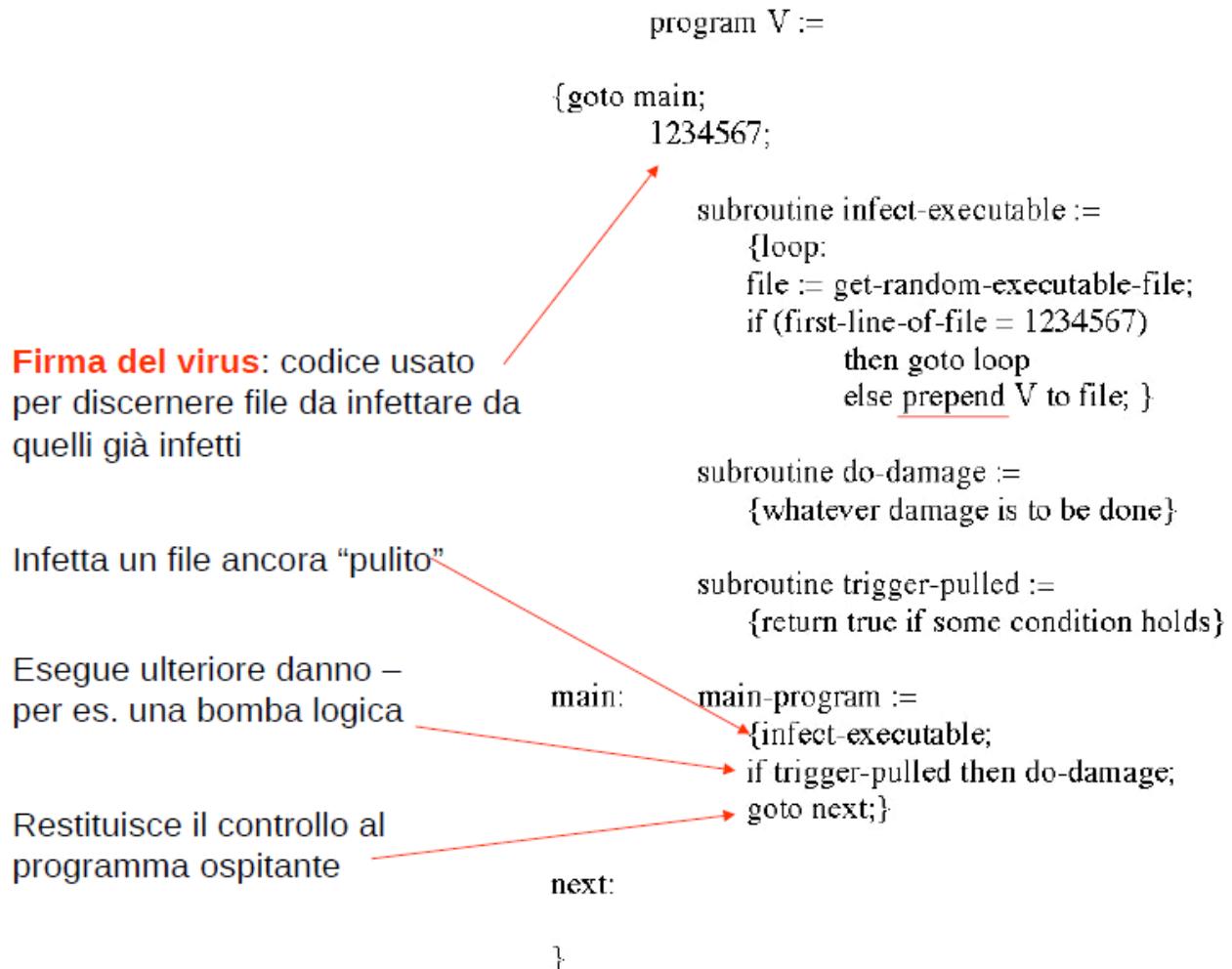
## Macrovirus

C'è stato un tempo, durante il quale, i virus più cattivi erano le macro di office. Le macro erano un modo per automatizzare operazioni. Con Visual Basic possiamo scriverle. Quindi una macro non è altro che un programma. WATA1 è stato scritto con macro VB per office.

## SW nocivo – distinzione delicata

	Tipo	Caratteristiche essenziali
Non replic. Replicano	Trapdoor	Concede accesso non-autorizzato
	Bomba logica	Si attiva solo su specifiche condizioni
	Cavallo di troia	Ha funzionalità illecite inattese
	Virus	Attacca un programma e si propaga attraverso qualunque mezzo, anche fisico
	Zombie	Usa macchina già violata per attaccare
	Worm	Viola macchine "ricorsivamente" attraverso la rete

# Struttura di un semplice virus



L'immagine mostra un semplice virus. Il numero in alto è la firma del virus. La firma del virus non bisogna confondere con la firma digitale. Questa firma è una stringa, un marcatore, che indica quali file ha già infettato il virus e quali no. Il riconoscimento di quel determinato virus si ottiene con quella firma.

Consideriamo il programma presente nell'immagine come programma imperativo. La sub-routine "infetta" fa questo: viene preso un file a caso e se già il virus ha visionato quel file, allora non fa niente e looppa, cioè non fa niente, altrimenti prependimi/appendimi in testa il virus. Poi c'è la sub-routine che indica che se c'è una certa condizione di attivazione verrà fatta qualcosa. Sembra una bomba logica ma il virus anche deve avere una condizione di attivazione per far eseguire il macrovirus. Infine c'è il programma principale che non fa altro che lanciare l'infezione, eseguire un danno e alla fine continua ad eseguire il programma ospitante (goto next).

Questi worm infetti da virus arrivano in modi diversi. Ad esempio per un periodo si propagavano come allegati eseguibili per la posta, per un periodo con i floppy, per un periodo con i boot sector delle pen drive, ecc.... .

## Dove inserire il virus in un file

Bisogna decidere dove inserire il virus. Un modo per rilevare il virus è controllare se all'inizio c'è la firma. Ma il virus potrebbe posizionarsi alla fine, quindi dovremmo controllare alla fine. Il virus potrebbe posizionarsi a metà e quindi dovremmo controllare a metà. Infine il virus potrebbe decidere un numero arbitrario di linee alla quale inserirsi. Il virus potrebbe avere delle caratteristiche di polimorfismo, potrebbe cambiare, potrebbe avere una strategia ciclica dove ogni volta cambia posto. L'antivirus dovrebbe fare tutti questi controlli e non è una cosa semplice da fare!

Qualcuno si è inventato che un modo facile per riscontrare la presenza di un virus era quello di controllare la dimensione di un file. Immaginiamo di monitorare un sistema e quindi di conoscere la dimensione ufficiale di una certa installazione, di una certa versione, ad esempio, di Word. Se conosciamo questo e poi andiamo a scansionare una installazione di Word e viene rilevata una dimensione più grande rispetto a quanto dovrebbe essere, allora possiamo segnalare un virus. Nel caso di aggiornamenti (che potrebbero alterare le dimensioni), l'antivirus dovrebbe essere consapevole di questo.

Se un modo di controllare la presenza di un virus è quella di controllare la dimensione, alcuni virus si comprimono in modo da nascondersi dal controllo sulla dimensione.

```
program CV :=  
  
1) Comprime il      {goto main;  
nuovo file da      01234567;  
infettare  
  
2) Appende il virus subroutine infect-executable :=  
all'inizio          {loop:  
  
3) Decomprime il      file := get-random-executable-file;  
resto del file       if (first-line-of-file = 01234567) then goto loop;  
ospite              (1) compress file;  
                    (2) prepend CV to file;  
                    }  
  
4) E lo esegue        main:  main-program :=  
main:                {if ask-permission then infect-executable;  
                     (3) uncompress rest-of-file;  
                     (4) run uncompressed file;}  
                     }
```

I vantaggi di questo metacodice, rispetto a quello precedente, è che realizza lo stesso obiettivo, però mantiene la dimensione del file risultante pari a quella del file originario senza il virus.

## Esempio di virus: Brain

Uno dei virus più storici è il virus Brian. Attacca sistemi operativi Microsoft. Rinomina il nome del disco in Brain e alcune versioni cancellano frammenti di memoria di massa. Si propaga.

La posizione 19 del vettore degli interrupt contiene l'indirizzo della procedura per gestire l'I/O. Il virus setta questo indirizzo al proprio indirizzo di memoria in modo tale che l'I/O lo gestisce lui. Quindi il principio di questo virus è quello di sostituirsi al controllore dell'I/O.

In questa prima versione il virus non crea danni. La storia dice che il virus controlla se il 5° e il 6° dei byte letti contengono la sua firma (il valore esadecimale 1234); se NO infetta 6 settori a caso, altrimenti STOP.

Il danno principale del virus Brain è che va a sostituirsi al controllore dell'I/O. Come payload possiamo non mettere niente, oppure possiamo marcare i settori infetti come "danneggiati", così che il S.O. non li usi più. Quindi la nostra macchina subirà un calo vertiginoso di memoria di massa.

## Rimozione di SW nocivo

Per rimuovere un malware possiamo considerare i seguenti strumenti:

### 1. Antivirus

- Metodo più comune ed in continua evoluzione
- Limitato dalla necessità di aggiornamenti

### 2. Sistemi immuni

- Prototipo IBM, tardi anni 90', sfrutta antivirus
- Un'intera struttura distribuita di prevenzione

### 3. Software sentinella

- Integrato col sistema operativo
- Blocca l'esecuzione di codice nocivo

# Antivirus

Gli antivirus eseguono tre compiti:

1. **Individuazione**: trova un file con un virus
2. **Identificazione**: stabilisce quale virus sia
3. **Eliminazione**: lo rimuove dal sistema
  - Può essere problematica in caso di virus polimorfi

A seconda di come vengano eseguiti i compiti 1 e 2, si distinguono quattro generazioni di antivirus

Se volessimo implementare un antivirus, quello che dovremmo fare è che esso scansioni tutti i file del sistema e controlla se all'interno sono presenti firme di virus. Questo è quello che fa un antivirus. Queste sono le caratteristiche degli **antivirus di prima generazione**.

**Prima generazione:** confronto con un DB

1. Scansione dei file alla ricerca di firme prese da un DB
2. Scansione di file di applicativi alla ricerca di variazioni nella loro lunghezza rispetto alle lunghezze standard prese da un DB

**Seconda generazione:** uso di euristiche

1. Ricerca di frammenti di codice spesso (statisticamente) associati a virus
  - Il ciclo di compressione di un CV (compression virus)
2. Ricerca di violazioni di integrità
  - Appendere un checksum ad ogni file di applicativi. Se il file cambia, il checksum dovrebbe cambiare. Ma è codificato con una chiave mantenuta altrove.

Quindi tutte caratteristiche, tutte discriminanti, per discernere da un file alterato da quello originale.

**Terza generazione:** ricerca di azioni illecite

- Programma sempre residente in memoria
- Identifica azioni "illecite"
  - Quando mai un utente aprirebbe 100 shell al minuto per scopi non nocivi?
  - Quando mai un applicativo tenterebbe di cancellare 10GB in una singola richiesta?
- Blocca un'azione illecita non appena sia rilevabile

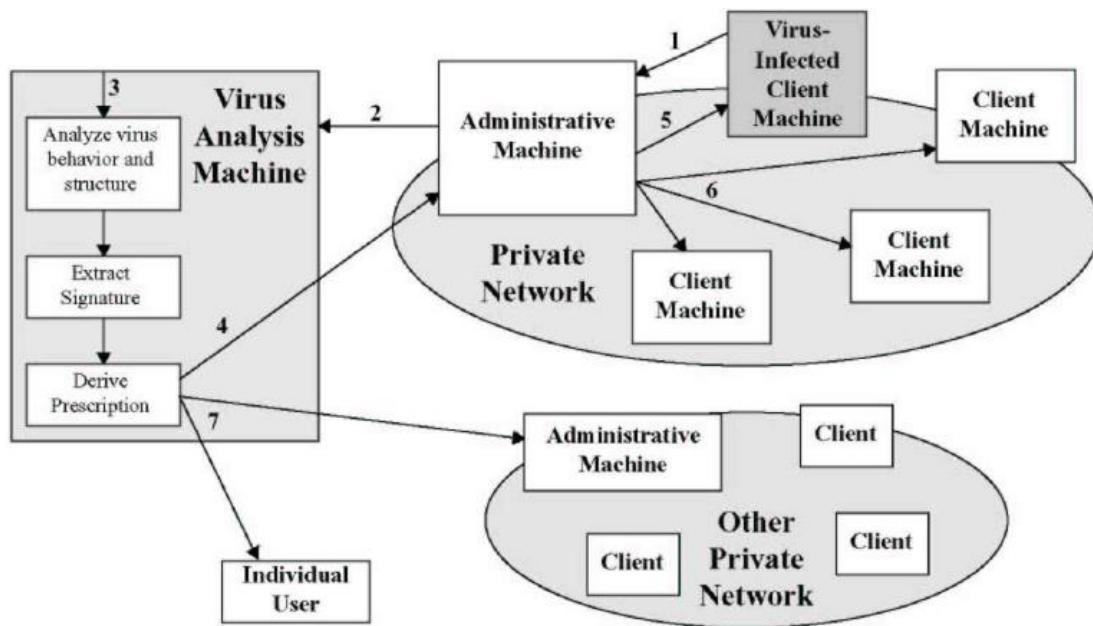
C'è stata un'epoca in cui gli antivirus erano dei programmi standalone, ovvero se volevamo lanciare il programma (l'antivirus) bene altrimenti niente. Se si dimenticava di lanciarlo non accadeva niente. Qualcuno pensò, che poiché l'utente è tonto, quindi potrebbe dimenticarsi di avviare il programma, di metterne uno che sta sempre residente in memoria e faccia la scansione frequentemente di alcune aree particolarmente soggette (ad attacchi di virus), come windows system. Furono questi i tempi in cui gli utenti si accorsero che gli si piantava il computer con l'antivirus. Allora alcuni non mettevano l'antivirus perché il computer gli si piantava. Quello era un tentativo mal riuscito di passare dalla seconda alla terza generazione, ovvero un programma sempre risiedente in memoria, che aveva queste caratteristiche di rilevamento di intrusioni.

**Quarta generazione:** insieme di tecniche. Quindi la possiamo considerare come un insieme di tutte queste generazioni viste. Gli antivirus di oggi sono antivirus di quarta generazione. Sostanzialmente quello che fanno lo sanno solo loro, perché sono programmi a codice chiuso, sono programmi commerciali.

- Usa un ventaglio delle prime tre tecniche
- Include capacità di ri-regolare il controllo d'accesso per limitare la propagazione di un virus trovato
  - *Se trova Melissa, impedisce a Internet Explorer di inviare dati e ad Outlook di inviare email*
  - *Trovato un qualunque virus, toglie i permessi di scrittura a qualunque utente*

# Sistemi immuni

I sistemi immuni sono un esercizio teorico. Sostanzialmente è un prototipo dell'IBM degli anni '90, che ha questo progetto:



Questo progetto ha una macchina che fa da antivirus centralizzata. Troviamo una bolla in alto a destra e quella in basso a destra e a sinistra c'è la macchina antivirus. Diciamo che è un modo per centralizzare l'antivirus, per renderlo una risorsa di rete.

Vengono chiamati sistemi immuni perché dovrebbe avere le caratteristiche di reagire in tempo pressoché reale. Quindi sarà amministrata da un amministratore specifico e non ci sarà bisogno dell'antivirus tradizionale. C'è un client per il sistema immune (Administrative Machine), il quale client è in comunicazione con la macchina centrale.

## Software sentinella

Un software sentinella, sostanzialmente, è un appendice del sistema operativo. Se abbiamo un programma benevolo o un antivirus, alla fine ambedue chiederanno di accedere alle risorse. Un software sentinella lo possiamo considerare come uno strumento che costruisce un ulteriore livello di filtraggio fra le richieste dei processi e la concessione delle risorse a questi processi. Quindi un software sentinella può decidere, in maniera centralizzata, se concedere queste risorse o meno (possibilmente chiedendo conferma all'utente).

## Punti deboli dei browser

I browser sono oggetti molto complessi. Essi eseguono codici PHP e Javascript. Le vulnerabilità che possono causare questi linguaggi sono enormi. Questo perché i browser sono processi che girano con tantissimi privilegi. Addirittura i browser, possono scrivere su store di sistema; semplicemente l'utente digita OK e il browser scriveva su uno store di sistema.

I punti deboli dei browser sono elencati in seguito:

- 1.** Gestisce il traffico del client
  - Indica almeno l'indirizzo di ritorno (IP)
- 2.** Gestisce le preferenze di sicurezza del client
  - Che ci faccio con le applet?
- 3.** Mantiene storia e cache delle pagine viste
  - Ho usato un terminale pubblico in aeroporto...
- 4.** Possiede le chiavi pubbliche di diverse RCA
  - Gestione delicata
- 5.** E' estremamente modulare
  - Auto installa continuamente nuovi plug-in
- 6.** Ha spesso accesso a tutte le risorse di sistema
  - Non possiamo "concedere" l'intero sistema a Internet!
- 7.** Memorizza password web dell'utente
  - Come le protegge?
- 8.** Esegue codice mobile

# **SSL**

## SSL

C'è chi chiama questo protocollo come SSL, c'è con TLS, e chi con HTTPS. HTTPS utilizza HTTP su SSL (oppure è l'utilizzo di SSL con HTTP). SSL è un protocollo di sicurezza e viene usato per fare canali sicuri. Quindi con HTTPS vengono trasportati ipertesti in maniera sicura. SSL è il protocollo più diffuso.

### **Nota**

HTTPS usa di default la porta 443 invece di 80 come HTTP.

SSL garantisce segretezza, integrità e autenticazione. Anche i protocolli giocattolo più o meno facevano la segretezza, l'integrità, l'autenticazione, ma con SSL è pratica (nella realtà usando questo protocollo otteniamo quanto detto prima).

Quindi con SSL raggiungiamo i tre obiettivi di livello base: segretezza, integrità e autenticazione.

## SSL versus TLS

TLS è l'ultima versione di SSL (versione SSL3.1). TSL è uno standard di internet.

## OpenSSL

**OpenSSL** è un'implementazione aperta e possiamo giocare con SSL.

- Implementa SSL/TLS
- Applicazione open source in costante sviluppo
- Permette di gestire certificati digitali
- Permette di ispezionare un'implementazione dei concetti che stiamo per vedere su SSL

### **OpenSSL: Richiesta certificati**

```
openssl req -new -newkey rsa:512 -nodes  
-keyout Key.pem -out Req.pem  
-config openssl.cnf
```

- **new**: indica nuova richiesta
- **newkey**: specifica formato chiave
- **noDes**: indica di salvare chiave privata in chiaro
- **keyout**: indica file con chiave privata
- **out**: indica file col certificato
- **config**: indica file con configurazioni di default

## OpenSSL: Rilascio certificati

```
openssl ca -policy policyAnything  
-out cert.perm -config openssl.cnf  
-infiles req.pem
```

- **ca**: opzione per la firma
- **policy**: indica politica da utilizzare per il rilascio
- **out**: indica file col certificato
- **config**: indica file con configurazioni di default
- **infiles**: indica file con la richiesta

## OpenSSL: Frammento del Client

```
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <netinet/in.h>  
#include <openssl/ssl.h>  
#define LEN 1024  
#define PORT 3000  
  
main(int argc, char**argv){  
    char CAfile[]="CACert.pem";  
    char buff_out[]="Prova di invio sicuro";  
    char buff_in[LEN];  
    SSL *ss;  
    SSL_CTX *ctx;  
    ...  
    ...
```

## Connessione versus sessione

Una sessione è un po' l'omologo di security association di IPSec. Quindi in una sessione ci sono le scelte fondamentali della sicurezza. Una sessione la possiamo realizzare attraverso più connessioni.

- **Connessione**: forma di trasporto per un determinato tipo di servizio
- **Sessione**: associazione fra un client e un server. Definisce i propri parametri di sicurezza
  - Simile ad SA in IPSec
- Una sessione può realizzarsi mediante più connessioni. Una connessione riguarda una ed una sola sessione

SSL è un protocollo complesso, è una suite di protocolli.

## Stato di una sessione SSL

Dentro una sessione; lo strato di una sessione; quali sono i campi che chiamiamo nel loro insieme una sessione, sono questi:

1. **Session identifier**: sequenza di byte arbitrariamente scelti dal server.
2. **Peer certificate**: certificato X.509 del nodo.
3. **Compression method**: specifica l'algoritmo di compressione applicato prima della codifica. SSL è una suite di protocolli e di fatto utilizza la compressione.
4. **CipherSuite**: specifica l'algoritmo di crittografia e l'hash usato.
5. **MasterSecret**: 48 byte condivisi da client/server.
6. **Is resumable**: è un flag e indica se vero se la sessione può essere ripristinata in nuove connessioni.

Il MasterSecret è l'oggetto fondamentale di SSL (verrà descritto più avanti).

## Stato di una connessione SSL

Per la connessione, troviamo questo:

1. **Server and client random**: sequenza di byte scelti da client e server per ciascuna connessione.
2. **Server MAC write secret**: chiave per MAC usata dal server.
3. **Client MAC write secret**: chiave per MAC usata dal client.
4. **Server write key**: chiave di codifica simmetrica usata dal server
5. **Client write key**: chiave di codifica simmetrica usata dal client
6. **Initialization vectors**: usato per inizializzare la codifica a blocchi
7. **Sequence numbers**: numeri sequenziali per i messaggi

Si tratta di punti limitati alla connessione. Ciò che sta scritta nella sessione era fondamentale, che verrà riutilizzato per ricostruire queste cose in una nuova connessione. È quindi logico che il MasterSecret sta nello stato sessione, perché è master, e poi sarà riutilizzato per la scelta dei parametri contingenti (come quelli scritti sopra), i quali costituiscono lo stato di una connessione. Quindi in sostanza si può ricostruire una connessione, il suo stato, a partire dalla sessione (dalle sessioni si derivano le connessioni). Per questo motivo il MasterSecret sta nella categoria “Stato di una sessione”.

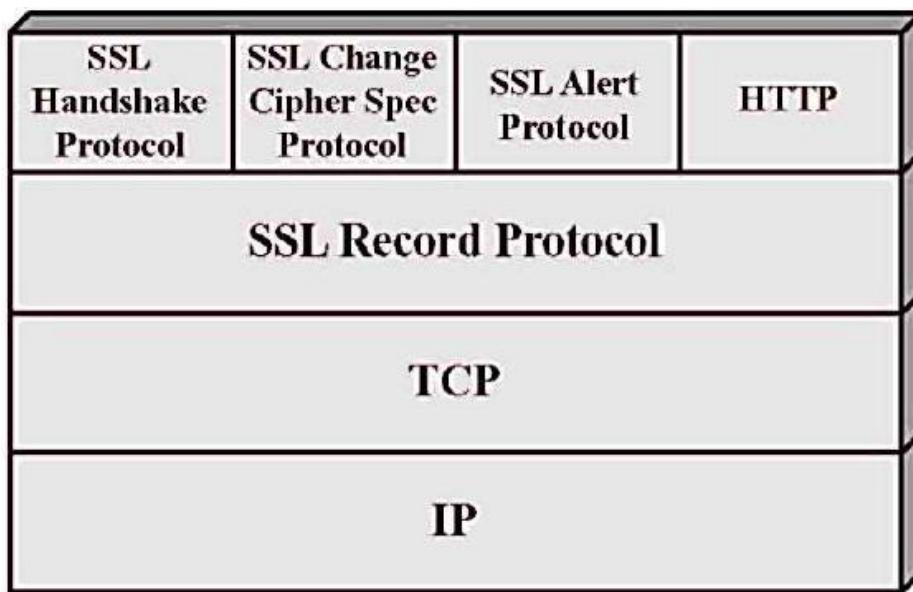
Dai punti 2 e 3 possiamo notare l'uso di chiavi simmetriche per il MAC. Abbiamo detto che il MAC lo possiamo considerare come un digest, oppure come un hash crittografico (per queste cose serviva una chiave). Server e client hanno ciascuno una chiave MAC. Questa chiave simmetrica deve essere condivisa. La specifica dice che, ciascuno utilizza la propria chiave in condivisione con l'altro. È come dire che ci sono due chiavi condivise fra i due, dove uno usa la sua chiave per spedire

il suo traffico, l'altro userà la sua chiave per spedire il suo traffico. Sono ambedue le chiavi condivise e quindi ambedue gli interlocutori possono codificare. Questo rafforza il valore di autenticazione, perché se io ho una chiave condivisa, il traffico con quella chiave o lo fatta io oppure il mio interlocutore. Invece con questa convenzione, che Server utilizzi la propria chiave condivisa con il Client per codificare il proprio traffico e il Client utilizza la propria chiave condivisa con il Server per codificare il proprio traffico, otteniamo anche una sorta di autenticazione e crittotesto, perché nessuno sa quale chiave stiamo usando per decodificare (se la chiave non è quella posso usarne un'altra, e così via.. diciamo che è un qualcosa un po discutibile!).

La chiave per il MAC è una chiave a breve termine per la codifica (chiave di codifica). (Una chiave di sessione è una terminologia generale per indicare una chiave breve).

## I protocolli in SSL

Dal punto di vista architetturale SSL è questo:



Il **record protocol** sta alla base. Come dice il nome, il record protocol, costruisce i record da dare in pasto a TCP/IP. Questi record come si fanno? Devono essere protetti, quindi ci saranno una o più chiavi, le quali vengono dal **protocollo SSL Handshake**. Quest'ultimo distribuisce le credenziali, le chiavi, sancisce le scelte crittografiche, sugella le scelte crittografiche. Poi troviamo il **protocollo SSL Change Cipher Spec**. Ed infine è presente il **protocollo SSL Alert**.

C'è ancora il dibattito sul fatto che SSL sia un protocollo di livello applicazione o no. Di fatto non è solo livello applicazione, perché il Record protocol sta a livello più basso, si interfaccia con TCP/IP. Il record protocol è quello che utilizza le primitive crittografiche per codificare record e li comprime pure.

## SSL Handshake Protocol (SSL HP)

Questa è la prima fase che viene fatta, ovvero vengono distribuiti i segreti in maniera segreta. Fatto questo c'è il bootstrap verso tutti gli altri protocolli.

Questo è il protocollo più complesso di SSL.

Viene stabilito il **MasterSecret** da cui creare altri segreti.

### SSL HP – formato dei messaggi

Il formato dei messaggi è codificato. Ecco il formato:

- **Type**: uno fra 10, (quelli riportati nella figura sottostante)
- **Length**: del messaggio in byte
- **Content**: parametri del messaggio.

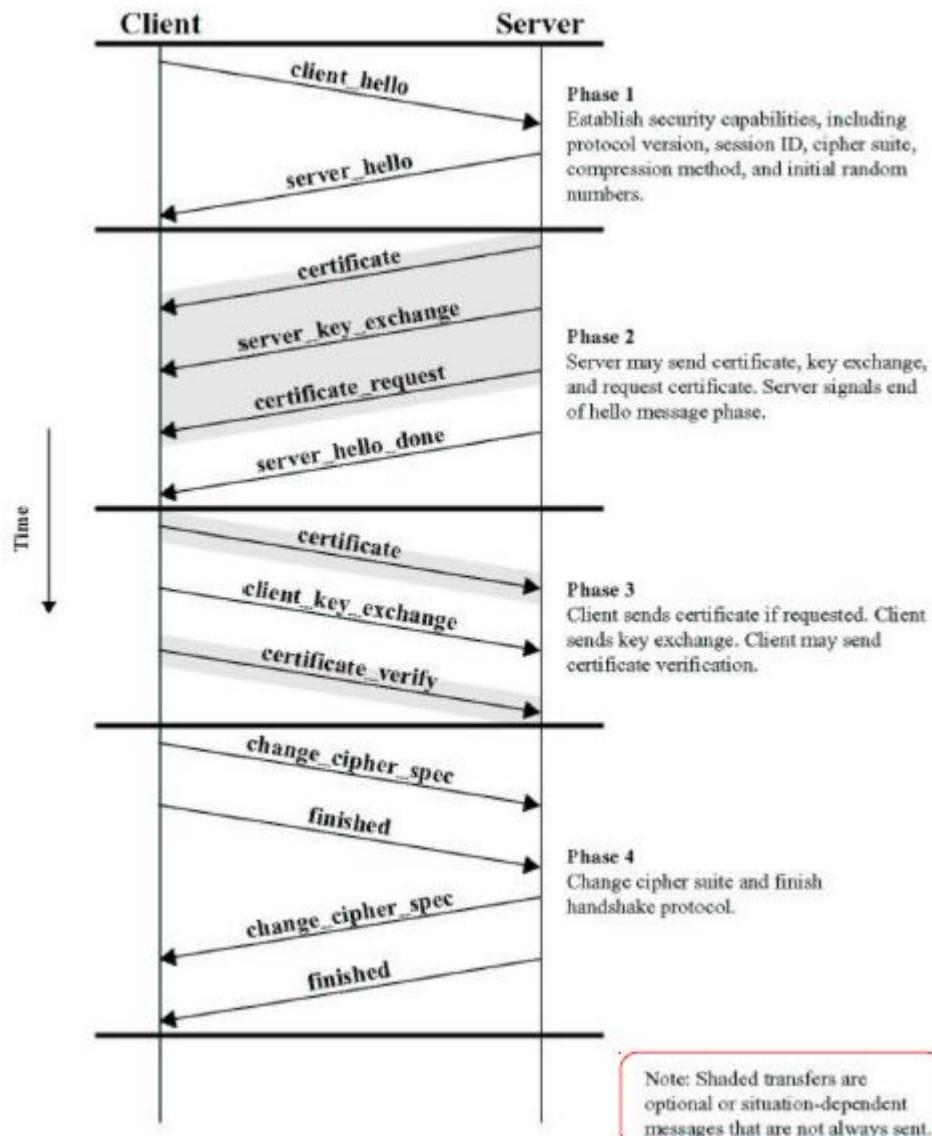
1 byte	3 bytes	$\geq 0$ bytes
Type	Length	Content

Handshake Protocol

### SSL Handshake Protocol Message Types

Message Type	Parameters
hello_request	null
client_hello	version, random, session id, cipher suite, compression method
server_hello	version, random, session id, cipher suite, compression method
certificate	chain of X.509v3 certificates
server_key_exchange	parameters, signature
certificate_request	type, authorities
server_done	null
certificate_verify	signature
client_key_exchange	parameters, signature
finished	hash value

## SSL HP



Questa presentazione di SSL mostra i nomi dei messaggi come vanno in relazione tra di loro. Notiamo pure le relative fasi. Alcune frecce hanno un “ombra grigia”. Quest’ultime indicano che sono opzionali. Dall’immagine vediamo quali messaggi appartengono a quali fasi.

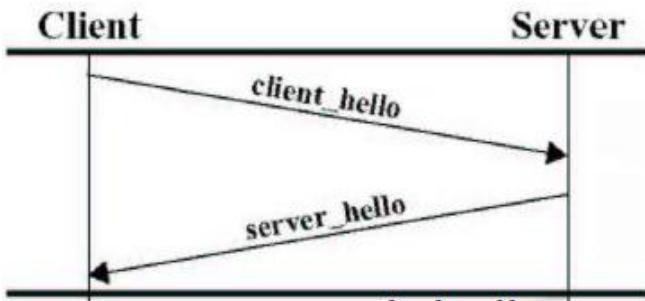
Nella **fase 1** c’è lo scambio di messaggi di tipo **hello** (`client_hello`, `server_hello`; descritti nella figura della pagina precedente). Il messaggio 0, ovvero il messaggio `hello_request`, non è presente perché è un messaggio superfluo.

La **fase 2/3** vedono un solo messaggio obbligatorio.

La **fase 4** c’è un messaggio che va dal Client al Server e viceversa, il messaggio **change\_cipher\_spec**. `change_cipher_spec` è lo stesso nome di uno dei protocolli della suite di SSL. Quindi potremmo dire che il protocollo `change_cipher_spec` è la parte finale dell’handshake protocol, ma sul piano formale è trattato come un protocollo normale.

Nelle fasi centrali si svolge una parte importante del protocollo: la parte di autentica, scambio di chiavi ed eventuali certificati (“eventuali” nel senso dei messaggi ombreggiati).

## SSL HP – Fase 1



Di seguito vengono dettagliati i parametri dei messaggi.

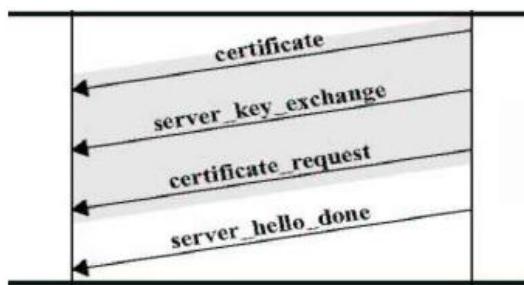
- Parametri di client\_hello
  - **Version**: più alta versione supportata dal client. Questo parametro è importante, perché Client e Server devono usare la versione del protocollo. Quindi non deve capitare che il Client usa la versione 3.0 e il Server la versione 3.1, perché altrimenti non comunicherebbero. (Possiamo dire che poiché è il Client che inizia è lui che decide la versione).
  - **Random**: timestamp (32 bit) + nonce (28 byte). Lo chiameremo **Client\_random** ed è un parametro fondamentale specialmente nel caso di ripristino. In realtà Client\_random è una mezza nonce, perché è una nonce potenziata con un valore temporale (abbiamo 28 byte di randomicità).
  - **Session ID**: uguale a (diverso da) 0 se il client vuole nuova connessione su nuova (medesima) sessione. Quindi significa che se Session ID è uguale a 0, il Client non vuole fare il ripristino della sessione, non vuole la sessione vecchia ma una nuova sessione. Se invece l'ID è diverso da 0, significa che il client vuole una nuova connessione su una vecchia sessione, ovvero sta ripristinando una vecchia sessione (infatti verrà inserito il Session ID della vecchia sessione).
  - **CipherSuite**: lista di preferenze crittografiche. Il client inizia e dice che tipi di protocolli usare, ad esempio RSA. Si fanno proposte, come tutti i protocolli di scambio, di negoziazione.
    - Quale protocollo di scambio chiavi, etc.
  - **Compression Method**: lista di algoritmi di compressione. Il client deve anche dire come comprimere il traffico, visto che SSL prevede anche la fase di compressione. Più che prevede, SSL, stabilisce, sancisce

questa fase, cioè la compressione fa parte della specifica SSL (si poteva fare anche a parte ma molto probabilmente sarebbe stata lasciata fuori dal protocollo, fatta in un altro modo, non protocollato).

I parametri, ovvero le componenti, di sever\_hello sono le stesse descritte prima, ovvero:

- Parametri di server\_hello
  - **Version**: minimo fra la proposta del client e la più alta supportata dal server. Se il Server supporta la versione 3.0 e il client manda versione 3.1, allora il Client deve adattarsi, ovvero deve pure il client scegliere la versione 3.0 perché altrimenti non ci sarà comunicazione.
  - **Random**: il server genera il proprio. Questo lo chiameremo **Server\_random** ed è fatto come descritto prima.
  - **Session ID**: se il client l'ha mandato diverso da zero, il server lo ricopia, altrimenti lo genera. Il client possibilmente ha mandato un Session ID diverso da zero e vuol dire che vuole ripristinare la sessione. Se il server ci sta, allora lo ricopia. Potrebbe anche decidere di non ricopiarlo. Se il Client ha mandato zero, vuol dire che non vuole ripristinare con una sessione nuova. In questo caso, il server, se ci sta a continuare il protocollo, genera un nuovo id di sessione. (questa spiegazione è plausibile per un accordo a due vie).
  - **CipherSuite**: scelta del server fra le proposte dal client
  - **Compression Method**: idem

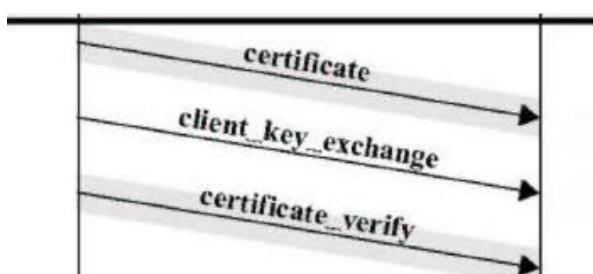
## SSL HP – Fase 2



- La fase 2 il Server manda la relativa lista di certificati, che possono agevolare la verifica della catena di fiducia.
- **server\_key\_exchange**: parametri pubblici, pars, firmati, per (variante di) Diffie-Hellman. Quando abbiamo parlato di DH non abbiamo parlato di firma, ma avevamo detto che c'era un attacco man in the middle, e il modo per risolvere questo attacco era di ibridare DH verso la crittografia asimmetrica. Questa idea (questa variante di DH) viene appunto da questo concetto (da concetto di server\_key\_exchange).

- **H(client\_hello.random,server\_hello.random,pars)**. Questi parametri pubblici vengono mandati, “attaccati” con client\_hello.random, server\_hello.random (i due parametri della fase precedente). Attaccati vuol dire in hash (H). Diciamo che questo è il digest della firma. Quindi, ricapitolando quest’ultimo concetto, nei parametri pubblici di DH ibrido, firmati in DH ibrido, firmati insieme ai due random della fase precedente.
  - Le porzioni nonce prevengono replay attack
  - **certificate\_request**: richiesta opzionale al client, con la lista di autorità accettate. certificate\_request è ombreggiato, quindi significa che il certificato del client, il server potrebbe non avercelo, ovvero significa che l’autenticazione del client con il server è opzionale. Quando ci colleghiamo ad esempio, con https al sito dell’Alitalia, il browser sta risolvendo il certificato per il server; mica l’utente dà al browser il proprio certificato. Quindi sul piano SSL il client rimane non autenticato.
- Ricapitolando:** il client è eseguito dentro il browser, ma se il client deve autenticarsi con il server cosa autentica? Esiste anche l’autenticazione del nodo. (in pratica l’autenticazione client verso il server si fa con login e password). Ricostruendo quanto detto fin ora: per l’autenticazione del server e client noi utenti vediamo solo https, quindi il client ha autenticato il server per noi tramite SSL. L’autenticazione del client con il server è opzionale. Cosa si intende per autenticazione del client con il server? Si intende l’autenticazione del nodo (di una macchina) o di un utente? Si intende l’autenticazione del nodo, a meno che non venga specificato User. Cmq è opzionale. Spesso l’autenticazione si fa al livello applicazione, tramite login e password. Significa che viene autenticato l’utente; tanto è vero che possiamo prenotare un biglietto aereo anche dal browser dell’aeroporto. In questo modo Alitalia autentica l’utente, non gli e ne può fregar tanto di autenticare il nodo. Però SSL prevede opzionalmente l’autenticazione client, client inteso come nodo. Quando quest’ultimo non viene fatto il problema è ovviato spostandolo al livello di autenticazione utente, quindi fuori SSL (perché è l’applicazione Alitalia che ha il database con tutti i dati e inseriamo all’applicazione le credenziali per accedere).
- **server\_hello\_done**: server OK, client verifica.

### SSL HP – Fase 3



- **certificate**: client manda il proprio certificato, oppure no\_certificate. Quindi opzionalmente il client manda la roba per autenticarsi.
- **client\_key\_exchange**: è la controparte di server\_key\_exchange
  - completa (variante di) protocollo DH (nel caso di DH la prima parte del protocollo iniziava nella seconda fase e termina in questa fase), oppure usa uno scambio RSA (questo nel caso in cui server\_key\_exchange della seconda fase non è stato effettuato. Quindi in caso di solo RSA serve solo questo messaggio) per mandare al server la **PreMasterSecret**.

Quindi se sia DH o scambio RSA, il server o se la calcola e la riceve la PreMasterSecret.

Alla fine della fase 3, abbiamo visto autenticazione con gli elementi di opzionalità e negoziazione o in un modo o nell'altro (DH o RSA) del PreMasterSecret. Il **MasterSecret** si otterrà algoritmicamente dal PreMasterSecret.

- **certificate\_verify** (verifica del certificato): hash del traffico visto (usa MasterSecret). Quest'hash di tutto il traffico è fatto così:

**H(MasterSecret,pad2, H(handshake\_messages,MasterSecret,pad1)) (\*)**

Ma ancora dobbiamo vedere come calcolare il MasterSecret dal PreMasterSecret.

Quindi **certificate\_verify** viene mandato dal client verso il server, e con questo messaggio il client sta dicendo al server l'(\*), dove troviamo il MasterSecret, il padding (quindi tutti i messaggi visti fin ora, con MasterSecret e padding). C'è del padding perché di fatto si devono soddisfare i requisiti di lunghezza di input richiesti dalle funzioni hash. Ma abbiamo pad1 in un posto e pad2 nell'altro. Quindi pad1 non necessariamente deve essere uguale a pad2 perché abbiamo i padding adatti per una parte e per l'altra. Nell'(\*) troviamo un hashing doppio, quest'ultimo di chiama **hashing ricorsivo** e rappresenta un tentativo di potenziare i limiti di invertibilità reali dell'hash.

- **handshake\_messages**: tutti i messaggi da client\_hello escluso certificate\_verify. Questi messaggi rappresentano quindi questo: prima di negoziare/calcolare il MasterSecret il client si mette al sicuro, ovvero si mette al sicuro mandando l'hashing di tutto quello calcolato fin ora. Ma l'hashing lo abbiamo usato per il controllo di integrità. Quindi mandando l'hashing di tutto quello visto fin ora, allora client e server si stanno scambiando la possibilità di controllare l'uno con l'altro, se ci siano stati in tutto lo scambio delle perdite. Ma l'attaccante potrebbe violare questo hashing? Ovvero potrebbe alterare il messaggio e poi applicare l'hashing per illudere il ricevente? Questa domanda con la

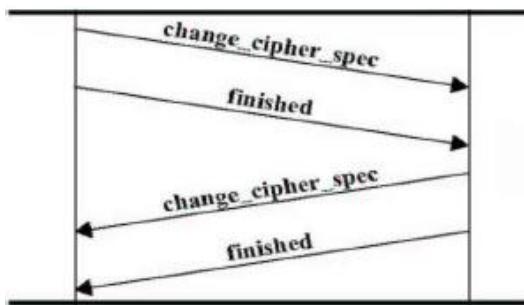
relativa risposta l'abbiamo vista quando abbiamo parlato di controllo di integrità. L'hashing è crittografico, ovvero c'è un parametro che si vede, il MasterSecret. Il fatto che ci siano scambi tramite DH o RSA e tutte le altre cose descritte prima, fa sì che in controllo di integrità sia effettivo, non sia violabile.

- Contro forme di replay attack
- Non esiste analogo messaggio dal server. Quindi il controllo di integrità in questa fase gli e lo da il client al server.

Quindi, ricapitolando, il protocollo di scambio di chiavi descritto prima fa questo:

- SSL ammette Diffie-Hellmann o sue varianti, oppure protocollo di scambio RSA per produrre PreMasterSecret
- Si espleta con i messaggi server\_key\_exchange e client\_key\_exchange
- Scambio RSA senza server\_key\_exchange
  1. Client → Server : {A, PreMasterSecret}Kb
  2. Client e Server calcolano offline MasterSecret=f(PreMasterSecret)

## SSL HP – Fase 4



In quest'ultima fase, client e server, fanno un controllo finale sull'integrità.

- **change\_cipher\_spec**: viene usato per attivare le scelte fatte della fase 1. Cioè da ora in poi usa le preferenze scelte crittografiche appena negoziate (e di queste oltre agli algoritmi di encryption, fa parte il PreMasterSecret).
- **finisched**: è questa concatenazione di hash

**MD5(MasterSecret,pad2),  
MD5(handshake\_messages, sender, MasterSecret,pad1)),  
SHA(MasterSecret,pad2),  
SHA(handshake\_messages, sender, MasterSecret,pad1)),**

Il senso di fare prima MD5 e poi SHA sta nel fare tutti i controlli di integrità sui messaggi visti fin ora. Si usano queste due funzioni hash diverse, per superare i limiti di invertibilità di ciascuna delle due funzioni. L'hashing è annidato per lo stesso motivo citato qualche pagina fa. Il MasterSecret è quello che lo rende crittografico. Il padding è quello necessario per applicare

le funzioni ( $\text{pad1} \neq \text{pad2}$ ). I messaggi di handshake sono tutti quelli che sono stati descritti fin ora. Ovviamente tutto questo fornisce l'integrità di sessione. Notare che il padding è codificato, perché se dobbiamo fare un controllo di integrità, prendiamo un hash e prendiamo i potenziali input dell'hash, applichiamo la funzione hash e facciamo il controllo per vedere se sono uguali (hash appena calcolato con l'hash che già abbiamo). Quindi per questo motivo il padding deve essere codificato, si deve conoscere, perché se mettessimo un padding differente non potremmo fare il controllo di integrità, il controllo di uguaglianza dei due digest. Quindi il padding, in un certo senso è un parametro pubblico. Non sta messo lì per rendere hashing più robusto, ma per farlo funzionare. Quello che rende più robusto quest'hashing è il MasterSecret (e ovviamente anche l'algoritmo di hash utilizzato).

## MasterSecret (MS)

Di seguito viene mostrato come calcolare il MasterSecret dal PreMasterSecret (PMS):

**MS =**

**MD5(PMS, SHA("A", PMS, ClientHello.random, ServerHello.random)),  
MD5(PMS, SHA("BB", PMS, ClientHello.random, ServerHello.random)),  
MD5(PMS, SHA("CCC", PMS, ClientHello.random, ServerHello.random)),  
...**

Prendo il PMS e attacchiamo SHA di ClientHello.random, ServerHello.random (che ambedue Server e Client conoscono perché se li sono scambiati nella fase 1), e scriviamo la lettera "A". "A" non sta per Alice, ma è semplicemente la stringa "A". Di quanto detto prima viene dato in pasto a MD5. Ci vuole il padding? Possibilmente sì, ma non ci interessa! Successivamente viene calcolato un secondo MD5 che verrà concatenato al primo. A differenza del primo troviamo "BB" che rappresenta la stringa "BB". Poiché nel primo è stato messo "A" e nel secondo "BB" ci siamo garantiti che i due MD5 sono diversi. Questo algoritmo verrà iterato fino a quando non avrò generato il numero di bit che mi servono, ovvero fino a 48 byte. Se il PMS è segreto, il MS è segreto? Sì. Se il MS è segreto, il PMS è segreto? Cioè si può ricavare il PMS se il MS è segreto? Dovremmo invertire l'hash, quindi la risposta è tanto no quanto è robusto l'hash.

## ... Riassumendo quanto detto fin ora ...

SSL si riassume come una suite di protocolli. Il protocollo più saliente è il protocollo di distribuzione delle chiavi che la terminologia ufficiale chiama handshake protocol. Ci sono anche altri protocolli che rendono operative le cose fatte dall'handshake protocol. Quest'ultime non sono altro che la distribuzione dei segreti fondamentali (questo viene fatto garantendo autenticazione e integrità). SSL fa questo con un protocollo di handshake fatto da 4 fasi. Il MasterSecret non cambia per tutto lo stato della sessione, ma ciò che cambia sono i segreti che vengono calcolati freschi sulla nuova connessione a partire dal MasterSecret congelato nello stato della sessione. È stato discusso precedentemente lo stato della sessione e della connessione. La sessione congela il MasterSecret, ovvero registra il MasterSecret, e quindi per tutta la sessione il MasterSecret non cambia. Su una nuova connessione che viene fatta sulla medesima sessione ripristinata, cambiano i segreti distribuiti.

Il PreMasterSecret viene concordato nelle fasi centrali del protocollo attraverso o lo scambio RSA (che prevede solo un passo in avanti) o DH potenziato con l'autenticazione che prevede due scambi. Dal PreMasterSecret viene calcolato il MasterSecret per mezzo delle seguenti operazioni:

(\*) MS =

MD5(PMS, SHA("A",PMS,ClientHello.random, ServerHello.random)),  
MD5(PMS, SHA("BB",PMS,ClientHello.random, ServerHello.random)),  
MD5(PMS, SHA("CCC",PMS,ClientHello.random, ServerHello.random)),  
...

Quest'operazione è una delle cose che rende SSL saliente, perché ci fa vedere l'espansione di un seme in un nuovo materiale crittografico, in altri termini di un segreto relativamente corto in modi segreti. La robustezza del segreto calcolato in questo modo sta nella robustezza della funzione hash.

Nella formula (\*), per calcolare il MS, troviamo delle lettere date in pasto alla funzione SHA, 'A', 'BB', 'CCC', che sono effettivamente scritte (vedere i prerequisiti delle funzioni hash). Il paramentro fondamentale è il PreMasterSecret arricchito con i parametri random del momento ClientHello.random, ServerHello.random. I puntini stanno ad indicare di continuare questo processo in espansione fino a quando ce ne bisogno, ovvero fino a quando non raggiungiamo 48 byte (se serve troncarli li trovi a blocchi di 48 byte).

**Osservazione:** Il MasterSecret è il segreto fondamentale. Non è che lui viene usato direttamente. Il MasterSecret viene generato dalla sessione e quindi comunque è un segreto a breve termine. Il MasterSecret caratterizza la sessione SSL. È costruito in una sessione SSL. Di fatto è una chiave di sessione. Nonostante si chiami chiave di sessione, il MasterSecret produrrà tutte le altri

chiavi. Quindi in un certo senso c'è una creazioni di chiavi a cascata. Queste creazioni vengono fatte in maniera computazionale. Computazionale vuol dire che queste chiavi sono costruite algoritmamente a partire dal MasterSecret.

Quindi abbiamo le due chiavi per il MAC, le due chiavi simmetriche e i vettori di inizializzazione. I vettori di inizializzazione sono fra gli input fondamentali di un qualunque algoritmo di encryption reale. Dopo che abbiamo tutta questa roba negoziata, la codifica dei dati SSL è simmetrica. L'utilizzo della crittografia asimmetrica è per lo più per ragioni della "busta digitale". Con la codifica asimmetrica otteniamo tante belle cose. La usiamo una volta, all'inizio, nella fase di bootstrap. Questo è il principio della busta digitale. Ricordiamo che il limite fondamentale della crittografia simmetrica è la condivisione del segreto iniziale, cosa che in uno scambio RSA viene risolto facilmente. La crittografia asimmetrica diciamo che è più lenta, è più pesante computazionalmente, quindi per questo motivo viene usata solo all'inizio e poi si va avanti con la crittografia simmetrica.

## Generazione delle chiavi

Quanto descritto prima sulla generazioni delle chiavi lo si può riassumere così:

- Dal MasterSecret vengono calcolati i segreti necessari
  - Client write MAC secret
  - Server write MAC secret
  - Client write key
  - Server write key
  - Initialization vectors
- Codifica dati (dopo SSL) e simmetrica, quindi veloce

## Generazione delle chiavi: key\_block

Il key\_block è il blocco delle chiavi. Noto il MasterSecret viene utilizzato lo schema seguente per costruire nuove chiavi:

(\*\*) **key\_block =**

```
MD5(MS, SHA("A",MS,ClientHello.random,ServerHello.random)),  
MD5(MS, SHA("BB",MS,ClientHello.random,ServerHello.random)),  
MD5(MS, SHA("CCC",MS,ClientHello.random,ServerHello.random)),  
...
```

A partite dal MasterSecret viene costruito un nuovo materiale crittografico.

Se il MasterSecret è congelato nello stato della sessione, la connessione nuova fa la fase 1. Nella fase 1 troviamo i due numeri random: ClientHello.random, ServerHello.random. Questi sono freschi, relativamente alla nuova connessione. Questa freshness verrà utilizzata per condire il MasterSecret congelato. Quindi in

questo modo viene creato un nuovo materiale crittografico per costruire i MAC secret e i write secret, freschi, relativi alla connessione attuale. Quindi malgrado il MasterSecret sia quello congelato, lo condiamo con qualcosa di nuovo, di diverso. Quanto detto lo possiamo riassumere così: possiamo costruire un segreto fresco malgrado il MasterSecret sia vecchio della sessione (congelato nella sessione), grazie ai numeri random che sono freschi e ovviamente anche alle proprietà hashing.

Quindi quanto scritto prima, descrive come si fa una chiave di sessione “qualsiasi” (ad ex. Client write MAC secret, e anche tutte le altri chiavi si fanno così (\*\*)), e tutto quello spiegato prima descrive come si realizzano queste chiavi!

## Ripristino di sessione

Se il flag di ripristino è “OK” allora si può fare il ripristino. Nella fase 1 il client può proporre un ID di sessione usato e poi spetta al server dire se ci sta o meno a ripristinare quella sessione. Questo schema è stato già spiegato precedentemente e si basava su accordo a due vie.

Il ripristino di sessione si fa nella fase 1 e nella fase 4. I due numeri random freschi, il calcolo del materiale crittografico sono nella fase 1 e lo schiocco di dita che dice d'ora in poi verranno usati (change\_cipher\_spec) accade invece nella fase 4. Naturalmente nel ripristino di sessione va mantenuto un ID di sessione ed è ovviamente una risorsa sensibile, perché identifica la sessione con le credenziali; quindi è come dire che tutto ciò che è relativo a quell'ID verrà accettato, lo scambio dei dati continuerà, ....

Ricevuto Session Id non nullo, il server lo ricerca nel proprio database e se lo trova e decide di accettare si fa il ripristino e si salta alla fase 4 con il completamento della costruzione dei nuovi key\_block a partire dal MasterSecret registrato.

## SSL Change Cipher Spec Protocol (SSL CCSP)

Il Change cipher spec protocol non è altro che un segnale di 1 byte.

Consiste in due soli messaggi: client e server si scambiano il byte per il valore 1. Più che un messaggio è un “OK” tra Client e Server per quello che si sono detti, perché non cita materiale saliente. Quello che si sono detti è una sorta di sincronizzazione. Cioè una volta fatto lo scambio di tutto, una volta che il dialogo è avvenuto allora verrà inviato questo segnale che indica che d'ora in avanti quello è il materiale crittografico che verrà usato. Questo è un protocollo ed è stato descritto nella fase 4.

## SSL Alert Protocol (SSL AP)

Nel protocollo di Alert:

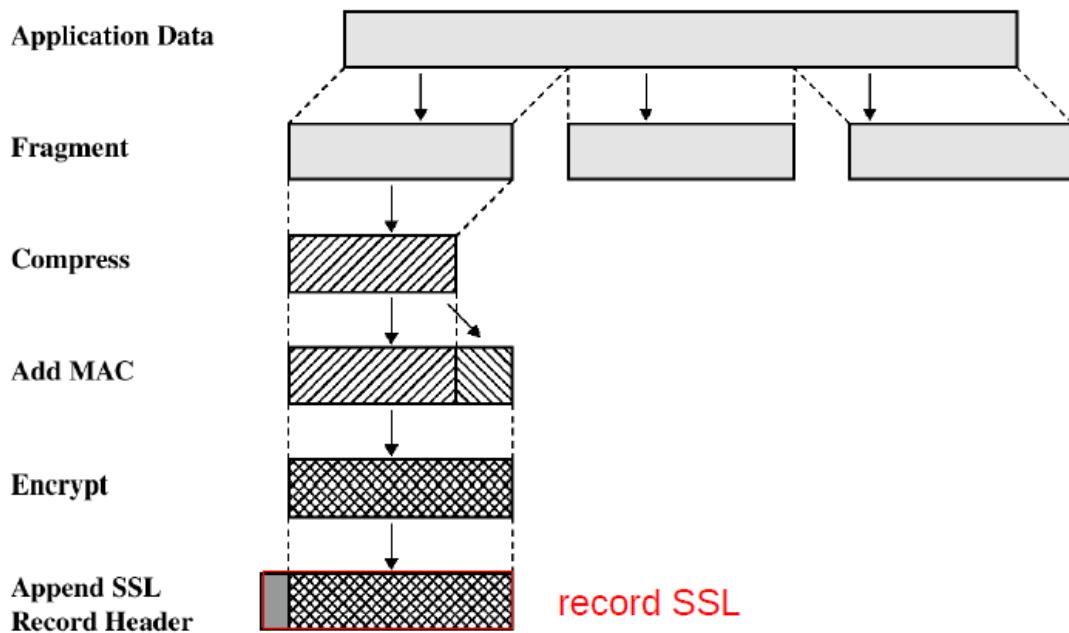
- Ogni messaggio composto da due byte
  - **Level**: livello d'allarme
  - **Alert**: quale allarme
- 2 livelli d'allarme. Gli alter possono essere fondamentali (fatal) oppure semplicemente dei warning.
  - **Fatal**: la connessione termina, non se ne possono aprire altre sulla sessione
  - **Warning**: problema risolvibile

Ecco alcuni alert:

- **SSL AP – fatal alerts**
  1. **unexpected\_message**: messaggio inatteso
  2. **bad\_record\_mac**: MAC incorretto
  3. **decompression\_failure**: la funzione di decompressione ha ricevuto input improprio
  4. **handshake\_failure**: il server non è riuscito a negoziare i parametri di sicurezza
  5. **illegal\_parameter**: un campo di un messaggio di handshake è inconsistente
- **SSL AP – warning alerts**
  1. **close\_notify**: notifica di chiusura connessione
  2. **no\_certificate**: certificato richiesto indisponibile
  3. **bad\_certificate**: certificato ricevuto corrotto
  4. **unsupported\_certificate**: tipo non supportato
  5. **certificate\_revoked**: revocato dall'autorità
  6. **certificate\_expired**: certificato scaduto
  7. **certificate\_unknown**: certificato sconosciuto

## SSL Record Protocol (SSL RP)

SSL dice come utilizzare il materiale crittografico proveniente dall'handshake protocol per proteggere i dati. Le fasi del record protocol sono i seguenti:



L'algoritmo fa questo:

- Si decompongono i dati
- Si comprimono
- Si aggiunge un pezzo di Autentication code
- Si codificano
- E infine si aggiunge il record SSL.

## SSL RP – funzionamento

Il funzionamento è l'algoritmo descritto prima. Riassumendo questo algoritmo abbiamo:

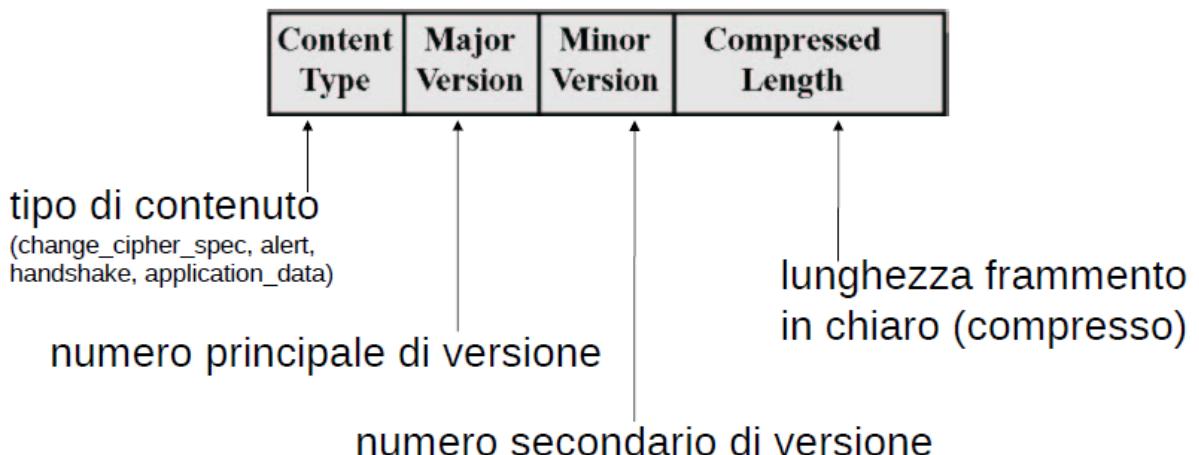
1. **Fragment**: ogni messaggio frammentato in blocchi di  $2^{14}$  byte
2. **Compress**: algoritmo di compressione opzionalmente applicato
  - Nemmeno TLS in realtà lo impone
3. **Add MAC**: MAC ricorsivo, un hash che include un altro hash
  - usa chiave condivisa MAC\_write\_secret ottenuta da livello superiore

Il MAC ricorsivo si calcola così:

```
H(MAC_write_secret, pad2,  
    H(MAC_write_secret, pad1, seq_num,  
        SSLCompressed.type,  
        SSLCompressed.length,  
        SSLCompressed.fragment))
```

Il MAC ricorsivo ha la solita struttura. Viene usata una funzione H e si vede che è crittografico dall'utilizzo della chiave crittografica preposta, ovvero il MAC\_write\_secret.

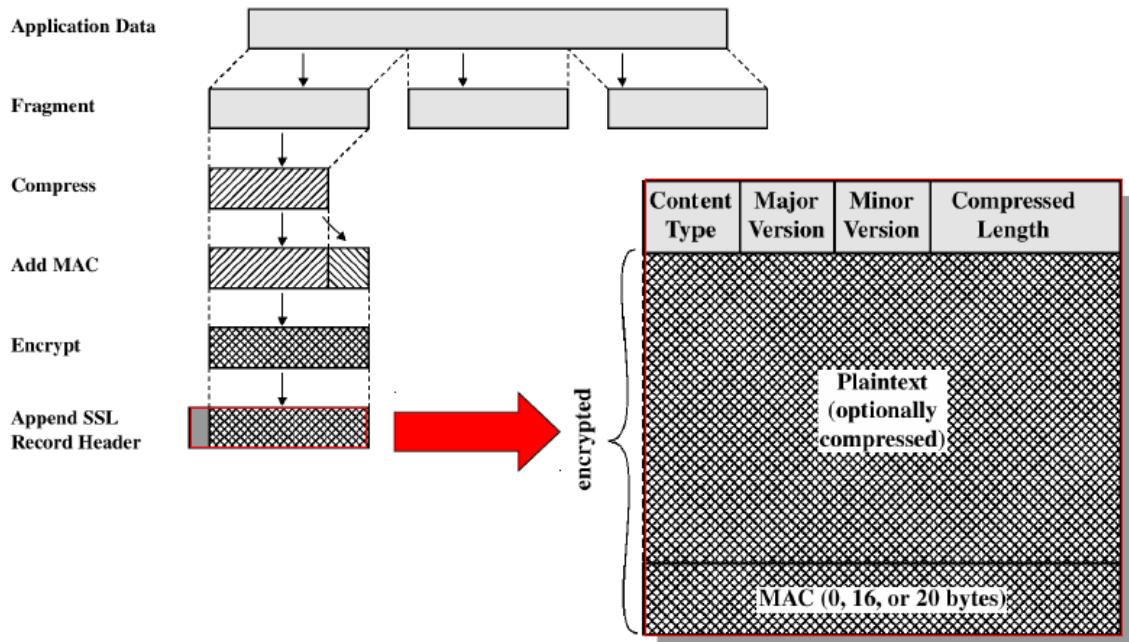
Di seguito viene mostrato una immagine che rappresenta l'SSL Record Header:



Il Major Version in questo caso noi stiamo considerando il '3'. Il Minor Version quello che indichiamo con .0 .1 ecc.. questi derivano dalla versione: SSL 3.0 / SSL 3.1, e così via.

TLS ha il numero di versione 3.1.

## SSL RP – formato del record



Quest'ultima immagine completa il record protocol. Si vede SSL è appeso al pacchetto criptato, si vede come è fatto l'header. A destra si vede l'intero formato del record.

# Transport Layer Security (TLS)

Con TLS la struttura del protocollo non cambia, però TLS è un tentativo di standardizzare SSL.

Rispetto a SSL, in TLS troviamo nuovi allarmi:

## ■ Nuovi allarmi fatal

1. **decryption\_failed**  
A TLSCiphertext decrypted in an invalid way: either it wasn't an even multiple of the block length or its padding values, when checked, weren't correct.
2. **record\_overflow**  
A TLSCiphertext record was received which had a length more than  $2^{14}+2048$  bytes, or a record decrypted to a TLSCompressed record with more than  $2^{14}+1024$  bytes. This message is always fatal.
3. **unknown\_ca**  
A valid certificate chain or partial chain was received, but the certificate was not accepted because the CA certificate could not be located or couldn't be matched with a known, trusted CA.
4. **access\_denied**  
A valid certificate was received, but when access control was applied, the sender decided not to proceed with negotiation.
5. **decode\_error**  
A message could not be decoded because some field was out of the specified range or the length of the message was incorrect.
6. **export\_restriction**  
A negotiation not in compliance with export restrictions was detected; for example, attempting to transfer a 1024 bit ephemeral RSA key for the RSA\_EXPORT handshake method.
7. **protocol\_version**  
The protocol version the client has attempted to negotiate is recognized, but not supported. (For example, old protocol versions might be avoided for security reasons).
8. **insufficient\_security**  
Returned instead of handshake\_failure when a negotiation has failed specifically because the server requires ciphers more secure than those supported by the client.
9. **internal\_error**  
An internal error unrelated to the peer or the correctness of the protocol makes it impossible to continue (such as a memory allocation failure).

## ■ Nuovi allarmi warning

1. **decrypt\_error**  
A handshake cryptographic operation failed, including being unable to correctly verify a signature, decrypt a key exchange, or validate a finished message.
2. **user\_canceled**  
This handshake is being canceled for some reason unrelated to a protocol failure. If the user cancels an operation after the handshake is complete, just closing the connection by sending a close\_notify is more appropriate. This alert should be followed by a close\_notify.
3. **no\_renegotiation**  
Sent by the client in response to a hello request or by the server in response to a client hello after initial handshaking. Either of these would normally lead to renegotiation; when that is not appropriate, the recipient should respond with this alert; at that point, the original requester can decide whether to proceed with the connection. One case where this would be appropriate would be where a server has spawned a process to satisfy a request; the process might receive security parameters (key length, authentication, etc.) at startup and it might be difficult to communicate changes to these parameters after that point.

TLS utilizza HMAC standard (hashing ricorsivo standard). La standardizzazione si vede dall'OR esclusivo. Mentre su SSL veniva usata la virgola (questa come concatenazione di messaggi, separatore di parametri di una funzione, ecc.. ), qui invece c'è l'OR esclusivo.

HMAC si calcola in questo modo:

$$\text{HMAC.H(M)} = \text{H}(\text{K} \oplus \text{pad2}, \text{H}(\text{K} \oplus \text{pad1}, \text{M})) \quad (***)$$

HMAC è un algoritmo che non sancisce l'uso di una specifica funzione hash. Quindi stiamo calcolando un HMAC fissata una certa funzione hash H su un messaggio M. Leggendo la (\*\*\* ) dall'interno verso l'esterno vediamo che la chiave

$K$  che lo rende crittografico viene messa in XOR col padding e poi viene concatenato il messaggio  $M$ . Il risultato di questo hash viene concatenato a  $K$  XOR nuovo padding e ricalcolato l'hash.

TLS usa una **PRF (pseudo random function)** per espandere in maniera sicura i segreti. Una PRF è una delle migliori sorgenti di randomicità. Parliamo di pseudorandomicità nella misura in cui questi numeri risultano molto imprevedibili.

## TLS: P.H (non è la PRF)

Quello che sta per essere descritto **non è la PRF**. Definiamo una nuova funzione **P**. Questa funzione fa un pò le veci dell'espansione vista in SSL. La funzione P fa questo:

$$\begin{aligned} P.H(secret, seed) = & \text{ HMAC.H(secret, A(1), seed),} \\ & \text{ HMAC.H(secret, A(2), seed),} \\ & \text{ HMAC.H(secret, A(3), seed),} \\ & \dots \end{aligned}$$

dove

$$A(0) = \text{seed}, A(i) = \text{HMAC.H(secret, A(i-1))}$$

La funzione P ( $P.H(secret, seed)$ ) prende il seme che vogliamo espandere ( $secret$ ) con un nuovo segreto fresco ( $seed$ ), e li mette in HMAC in questo modo:

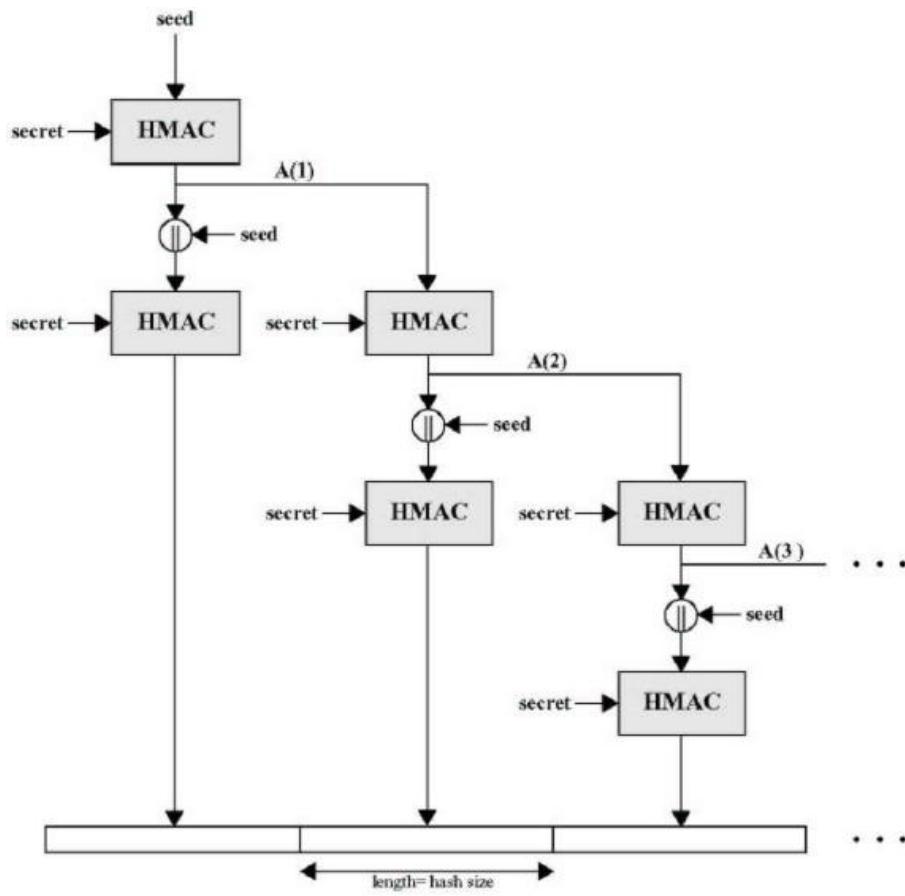
$$\text{HMAC.H(secret, A(1), seed)}$$

Questo ha una discreta forma di ricorsività.  $A(0)$  è il seme.  $A(1)$  è costruito come HMAC sul segreto e l'elemento precedente di A.  $A(1)$  è a sua volta inserito nell'HMAC.H. Quindi il vettore A indica la ricorsività dell'applicazione dell'HMAC. Se per esempio consideriamo  $A(1)$ , esso contiene l'applicazione dell'HMAC, però esso stesso è parametro per l'applicazione di una nuova HMAC esterna.

Quindi la differenza tra questo e SSL sta sostanzialmente nell'utilizzo dell'HMAC. Anche qui si può iterare un altro processo per generare un nuovo materiale crittografico.

Questo algoritmo non è la PRF. La PRF fa un ulteriore mescolamento sulla P.H. L'immagine della pagina successiva, mostra P.H raccontata in maniera grafica. All'inizio l'HMAC prende seed e secret, producendo  $A(1)$ . L'HMAC che sta in basso nella prima colonna in basso a sinistra, prende il risultato ( $A(1)$ ) insieme al segreto e va in output. Questa è la rappresentazione grafica della prima linea specificata prima. Allo stesso modo si procede con le altre. ....

Quanto descritto NON E' LA PRF.



## TLS: PRF

La PRF fa l'XOR di una duplice applicazione della P.H. duplice vuol dire che una volta faccio la P.H con  $H = \text{MD5}$  e poi la metto in un OR esclusivo con la P.H con  $H = \text{funzione hash SHA1}$ . Malgrado SHA1 e MD5 sono considerate al giorno d'oggi funzioni hash deboli, accade che questo utilizzo nel P e poi nella PRF non è ancora stato violato. Quindi empiricamente è sicuro.

$$\begin{aligned} \text{PRF(secret,label,seed)} &= \text{P.MD5}(s1, \text{label}, \text{seed}) \\ &\oplus \text{P.SHA-1}(s2, \text{label}, \text{seed}) \end{aligned}$$

## TLS1.0 versus SSL3.0

- certificate\_verify: rimosso MS e padding
  - $H(H(\text{handshake\_messages}))$
- finished: trasporta un hash diverso
  - $\text{PRF}(\text{MS}, \text{"finished"},$   
 $\text{MD5}(\text{handshake\_messages}),$   
 $\text{SHA-1}(\text{handshake\_messages}))$
- Il terzo parametro di PRF è la concatenazione dei due hash

Sostanzialmente arriviamo all'utilizzo della PRF nell'espandere il PreMasterSecret nel MasterSecret. Il MasterSecret viene calcolato nel seguente modo:

**MS1 = PRF(PMS, “master\_secret”, ClientHello.random, ServerHello.random)**

**MS2 = PRF(MS1, “master\_secret”, ClientHello.random, ServerHello.random)**

...

**MS = MS1, MS2, ...**

Algoritmo iterato fino ad ottenere 48 byte.

Quindi TLS per costruire il MS usa la PRF e la PRF si costruisce come descritto prima. Ovviamente anche qui notiamo la presenza dei due numeri random freschi provenienti dalla fase 1. È chiaro che se la PRF la utilizzo per calcolare il MS dal PMS, verrà utilizzata anche per calcolare i nuovi blocchi crittografici a partire dal MS.

Quindi banalmente potremmo dire che la differenza tra i due protocolli è l'utilizzo di HMAC standard e PRF in TLS rispetto a SSL.

I bit per le varie chiavi ottenuti con algoritmo analogo ma non identico

**key\_block1 = PRF(MS, “key\_expansion”,  
ClientHello.random, ServerHello.random)**

**key\_block2 = PRF(key\_block1, “key\_expansion”,  
ClientHello.random, ServerHello.random)**

...

Notare che qui c'è la stessa stringa data in pasto alla PRF, la stringa “key\_expansion”. In SSL non c'era sempre la stessa stringa. In questa notazione, il MasterSecret viene data nel primo blocco. Al secondo blocco diamo in pasto il blocco precedente.

## Concludendo

SSL per quanto sia il protocollo diffuso rimane un Key distribution protocol, sostanzialmente distribuisce chiavi in maniera segreta.

Fare compravendite è un qualcosa di più complesso.