

Sicurezza delle Reti

Marco Colognese

Ottobre 2018

Indice

Introduzione	3
Sicurezza Informatica	3
Privacy in Network Security	4
Anonymity Network	5
Algoritmi e Meccanismi di Crittografia	7
Simmetric Key Encryption	7
Placement of Encryption	8
Key Distribution	8
Algoritmi di Crittografia a Chiave Pubblica	9
Algoritmo RSA	10
Algoritmo di Diffie-Hellman	11
ElGamal (Variante di Diffie-Hellman)	12
Massey-Omura	12
Integrità dei messaggi e hash crittografici	13
Hash	13
MAC	13
Firma digitale	13
Public-Key Infrastructures	15
Certificati	15
X.509	15
Trust Models	16
Web of Trust: PGP	16
Network Security and Internet Protocols	17
Costruzione di un Protocollo per lo Sambio di Chiavi	17
Il Protocollo di Needham-Schroeder a Chiave Pubblica (NSPK)	19
Il Protocollo NSL (Needham-Schroeder-Lowe)	19
Tipi di Attacco	19
Il Protocollo di Otway-Rees	20
Il Protocollo di Andrew Secure RPC	20
Scambio delle chiavi con CA (Denning & Sacco)	21
Esempi di Attacchi	21
Kerberos, SSL, IPsec	22
Kerberos	22
SSL (Secure Socket Layer)	24
IPsec	26
IKE	28
Intrusion Detection System	31
Firewall	33

Introduzione

I sistemi (oggetti matematici) vengono costruiti con metodi formali basati su matematica e logica, migliorandone la loro **sicurezza**. La nostra *information society* è caratterizzata dall'espansione di internet nel quale convergono le informazioni e comunicazioni. Le infrastrutture si basano quasi tutte su sistemi informativi connessi alla rete. La sicurezza è un ambito interdisciplinare e conferisce potere ad un'organizzazione.

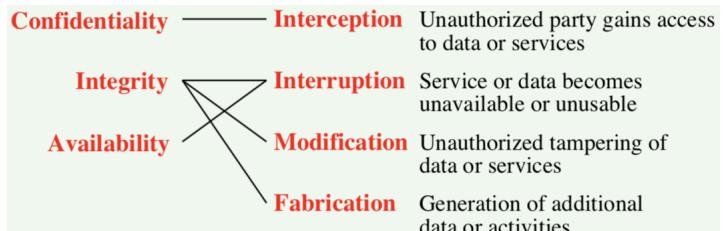
Sicurezza Informatica

- **Computer Security:** ha a che fare con la prevenzione e la scoperta di azioni non autorizzate da utenti per mezzo di un sistema di computer. Si basa principalmente sul concetto di *autorizzazione* e *security policy*.
- **Network Security:** si basa su un'infrastruttura di rete e politiche adottate dal suo amministratore. Si vuole proteggere la rete e le sue risorse da accessi non autorizzati.
- **Information Security:** riguarda le informazioni, ricavabili dai dati, indipendentemente dai computer. Si occupa di proteggere l'informazione e i sistemi informativi da accessi, usi, modifiche o distruzioni non autorizzate.
- **Security policy:** indica che quali comportamenti del sistema sono ammessi o meno. Viene formulata per raggiungere delle *security properties* (oppure *security goals*) come la più nota **CIA**:
 - **Confidentiality (Secrecy):** evitare che l'informazione venga rivelata a terzi;
 - **Integrity:** evitare modifiche da parte di terzi;
 - **Availability:** evitare problemi di funzionalità/servizio (es. *DoS attack*).
 - **Accountability:** per tenere traccia dei responsabili delle azioni tramite log che vengono poi spediti ad un altro server online per evitare che vengano corrotti quando il sistema viene attaccato;
 - **Authentication:** per verificare l'identità degli utenti, l'origine dei dati e permettere un *access control*. Può essere effettuata con qualcosa che si ha (card), che si conosce (password) o che si è (firma, impronta digitale);
- **Security mechanisms:** per far rispettare la policy del sistema (in tutti gli *ambienti malevoli*).

Protezione di un Sistema

- *Prevention:* si vogliono prevenire brecce del sistema con opportune tecnologie di sicurezza e difesa (es. firewall);
- *Detection:* in caso di attacco, è necessario individuarlo attraverso *intrusion detection systems* (es. file di log);
- *Response:* in caso di breccia, è necessario avere dei metodi per il recupero degli assets attraverso dei backup e interpellando le forze dell'ordine.

Quando si lavora per la sicurezza di un sistema, vanno prese contromisure per effettuare **risk minimization**, limitando il più possibile le vulnerabilità ad un valore accettabile, per mettere al sicuro gli assets da potenziali furti.



Una **vulnerabilità** è una debolezza del sistema che può essere sfruttata da un *attacco* per creare un danno. Esse emergono quando si ha a che fare con un ambiente ostile: pc in mani sbagliate, internet, software vari.

L'Analisi del Rischio e la sua riduzione si divide in passi:

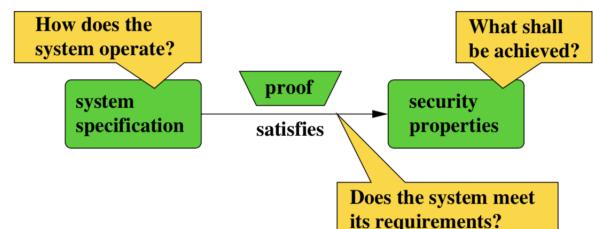
- analisi dei rischi esistenti: identificare gli assets da proteggere e i rischi ai quali sono esposti;
- analisi delle soluzioni di sicurezza: quale contromisura riduce i rischi, valutando anche i rischi che porterà.

Il Managing Security prevede alcuni passi per implementare una soluzione:

- *security analysis*: indaga sui possibili attacchi che possono mettere a rischio gli assets, proponendo una policy e soluzioni con un costo appropriato.
- *threat model*: documenta i possibili attacchi ad un sistema, immaginando tutte le vulnerabilità sfruttabili.
- *risk assessment*: studia la probabilità di un attacco nell'ambiente del sistema e gli assegna un costo (trova il rischio).
- *security policy*: elenca i possibili attacchi e le rispettive contromisure possibili. Si analizzano i costi delle contromisure in relazione al rischio per ottenere un compromesso valido.
- *security solution*: vengono distribuite le tecnologie appropriate al costo adeguato; è un lavoro di bilancio ma è giusto fare gli sforzi giusti dove necessario.

Formal Methods

Si costruisce un modello formale dei comportamenti del sistema (astrazione del programma concreto). Vengono formalizzati i requisiti come proprietà di sicurezza e si verifica che il sistema le soddisfi. Bisogna documentare come deve operare il sistema, i requisiti di sicurezza in modo non ambiguo e validare il sistema tenendo conto dei requisiti.



Privacy in Network Security

La *confidenzialità* riguarda l'introduzione di un *access control* per evitare che certe informazioni vengano lette da persone non autorizzate. Spesso con *privacy* si intende l'*anonimità*: mantenere privata l'identità.

- *Privacy*: scegliere cosa gli altri possono sapere (confidenzialità delle informazioni non si vogliono condividere).
- *Data Protection*: assicurare che i nostri dati non siano distribuiti e usati in modi indesiderati (si segue una *policy*).
- *Anonymity*: condizione nella quale la vera identità non è conosciuta (confidenzialità della propria identità).
- *Inosservabilità* delle azioni nel momento in cui vengono effettuate.

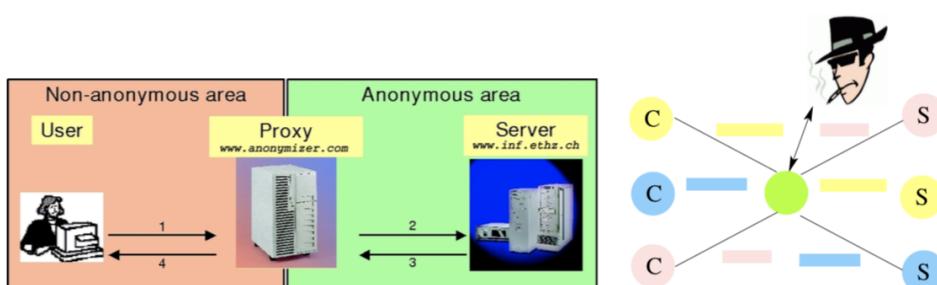
L'*anonimità* è difficile da ottenere in una rete pubblica poiché gli headers dei pacchetti identificano i destinatari e nella rete è possibile effettuare *traffic analysis* (servono tecnologie per contrastarla).

Nonostante il payload sia criptato, è comunque visibile ed è possibile violare l'anonymità.

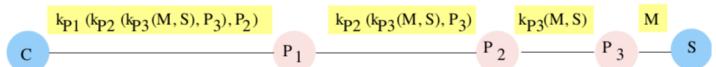
Anonymity Set: l'unico livello di anonimia in rete è dato dal fatto che le *azioni* di una persona possono essere osservate ma *non distinte* da quelle di altri. Questo gruppo è chiamato *anonymity set* (più è grande, meglio è).

L'obiettivo è rendere anonimi mittenti e destinatari durante le comunicazioni per dare confidenzialità alle identità.

- *ricevitore*: reso anonimo quando si spediscono messaggi in broadcast o multicast (nell'anonymity group).
- *mittente*: i pacchetti (es. le richieste *HTTP*), vengono anonimizzate dai server proxy. Il **problema** è che il *proxy* conosce tutto (si può fare *traffic analysis*). Le **soluzioni** sono catene di proxy in cascata, *mix networks* ...



Catene di proxy in cascata: ogni proxy conosce solo il *next hop*. È ancora possibile effettuare *traffic analysis*.

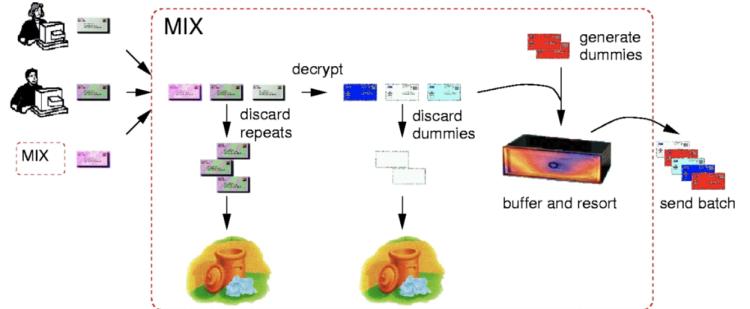


Mix Networks: è un meccanismo per la creazione di un *canale anonimo*. È progettato per lavorare in un ambiente con un hacker attivo. Originariamente era stato proposto per sistemi di email anonime.

Un **Mix** è un server che processa dei messaggi per *B*.

- l'input del mix è la parte di sinistra;
- il Mix genera in output la parte di destra
- R_i sono blocchi di stringhe di bit randomici.

Questa è una variante di un *single-proxy*. Il Mix esegue anche altre operazioni per sconfiggere l'analisi del traffico.

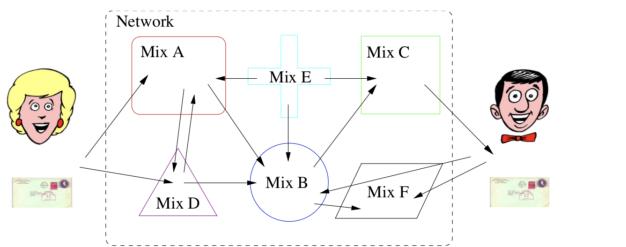


I Mix lavorano con oggetti di dimensione uniforme (messaggi divisi in blocchi di dimensione fissa).

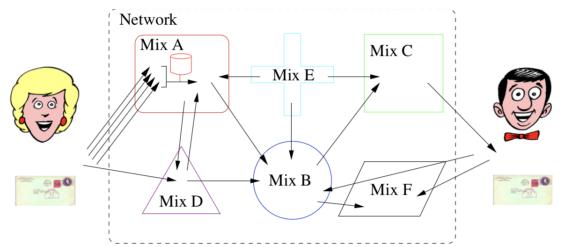
Ogni **Mix** riceve in input tanti messaggi da origini diverse (è meglio quando ci sono tante sorgenti), scarta i pacchetti ripetuti (per evitare i *replay attack*), li decrypta, scarta i pacchetti fittizzi, ne genera di nuovi (perché magari ci sono pochi utenti ed è necessario prevenire che i messaggi vengano tracciati), ne scambia l'ordine e li manda al *next hop*. Un *single-mix* ha le stesse debolezze di un *single-proxy*. È più sicuro averne tanti (in domini amministrativi diversi). Se uno o più Mix sono sotto il controllo di un hacker, è sufficiente che ne funzioni correttamente almeno uno.

Con questo funzionamento possiamo affermare che le **mix networks garantiscono anonimità**. Non c'è correlazione tra input e output del mix. Il **mittente resta anonimo** anche al destinatario.

Svantaggi: nelle reti molto trafficate ci possono essere ritardi; è sufficiente ridurre o eliminare i pacchetti *dummy*.



Con i *dummy message* è difficile tracciare la conversazione.



Il *replay filter* previene i *replay attack*.

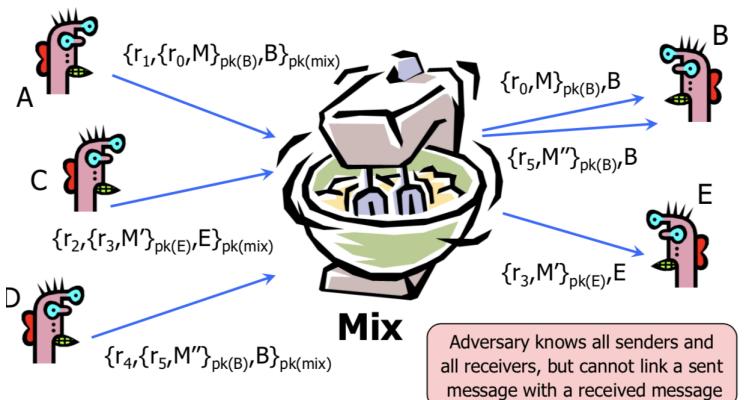
Per **rispondere** ad un *mittente anonimo*, esso deve includere $\langle\langle R_1, \langle R_2, \dots, \langle R_n, A_x \rangle_{K_n} \dots \rangle_{K_2} \rangle_{K_1}, K_x \rangle$ nel messaggio. Il destinatario non potrà risalire all'IP di *A* poiché resterà sempre crittato da stringhe randomiche K_n scelte da *A*. Il risultato finale ritornato ad *A* sarà: $\langle A_x, \langle \langle \langle R_0, M \rangle_{K_x} \rangle_{R_1} \rangle_{R_2} \dots \rangle_{R_n} \rangle$.

Anonymity Network

Internet è stato progettato come una **rete pubblica** e le informazioni di routing sono pubbliche (un ascoltatore passivo può sapere se due utenti stanno comunicando). La cifratura non nasconde le identità nei *packet-headers*.

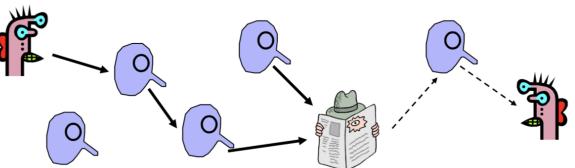
L'**Anonimità** è fondamentale nei sistemi utilizzati per certi scopi come: votazioni online, comunicazioni segrete, sistemi governativi o militari ...

Permette di rendere un utente impossibile da identificare in mezzo ad altri utenti.



Le *Mix Networks* hanno una latenza eccessiva dovuta alla cifratura e decifratura a chiave pubblica (computazionalmente costosa). Questo ritardo è accettabile in un servizio di mail anonime ma non in *web browser* anonimo.

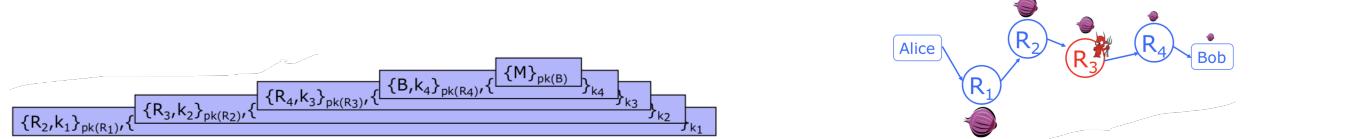
Un'idea può essere il **randomized routing** (utilizzato anche dall'*Onion Routing*) che nasconde la sorgente di un messaggio attraverso un routing randomico. In questo modo i router non sanno se la sorgente apparente del messaggio è davvero il mittente oppure un altro router.



Onion Routing

Il mittente sceglie una sequenza randomica di router (alcuni possono essere sotto il controllo dell'hacker). Sceglie anche la lunghezza del percorso.

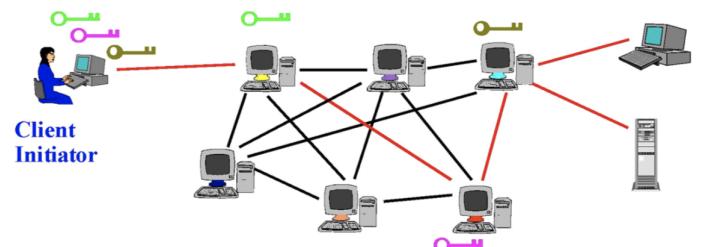
Le informazioni di routing sono criptate con la *public-key* del router che conosce solamente l'identità del successivo.



Tor

Tor è una *onion routing network* di seconda generazione. Progettato per *web browsing* anonimo a bassa latenza.

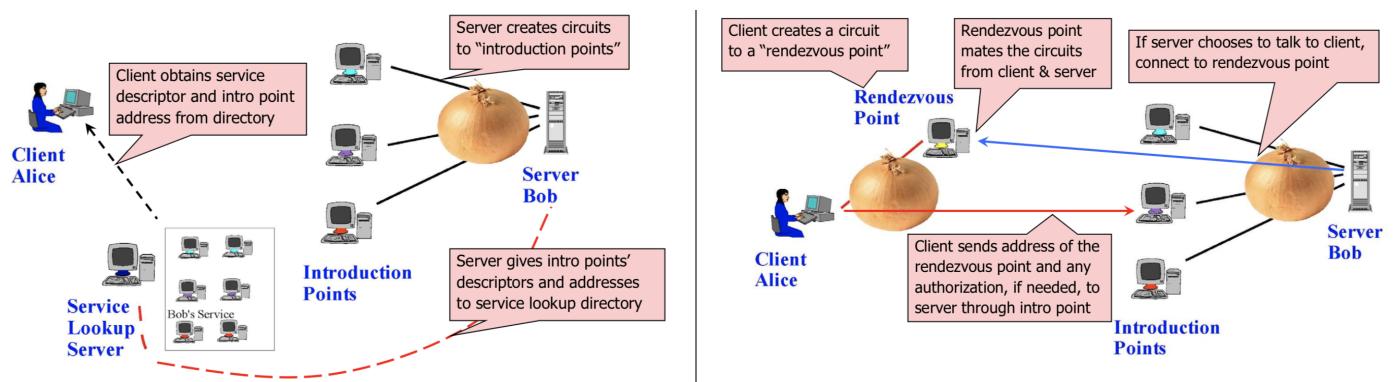
- Il *client-proxy* stabilisce *session-key* simmetrica ed un circuito con gli *Onion Router* #1 #2 #3 (tunnel).
- le applicazioni client si connettono e comunicano sul circuito *Tor* stabilito.
- i datagrammi vengono decriptati e re-criptati ad ogni collegamento.



Problemi di gestione:

- molte applicazioni possono avviare troppe *TCP* stream su un unico circuito;
- i router di *Tor* non hanno bisogno di *root-privileges*, incoraggiando le persone a settare i propri router (più partecipanti, più anonimità);
- *directory server*: per tenere traccia degli onion router attivi, le loro locazioni e chiavi pubbliche, controllare i nuovi router che accedono alla rete (per evitare *Sybil attack* in cui un attaccante crea tanti router), inviare la chiave del server con un codice *Tor*.

Hidden Server: l'obiettivo è mettere online un server accessibile a tutti, ma che nessuno sappia dove si trova. Deve resistere alle censure, ai *DoS attack* e non deve essere trovato fisicamente.



Algoritmi e Meccanismi di Crittografia

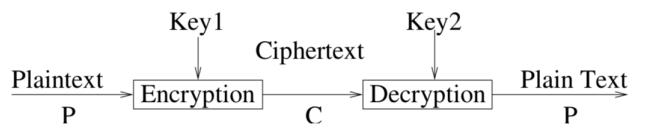
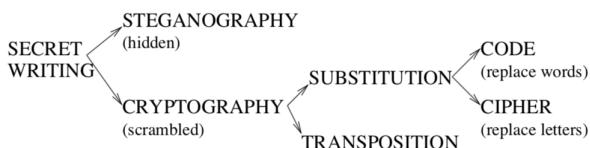
Un canale di comunicazione non affidabile può diventare sicuro applicando le 3 proprietà tramite la **Crittografia**:

- *Confidentiality*: le informazioni trasmesse devono restare segrete;
- *Integrity*: le informazioni non devono essere corrotte, alterate o eliminate;
- *Authentication*: gli utenti devono sapere chi si trova dall'altro lato.

Per criptare un messaggio si possono utilizzare algoritmi diversi:

- **algoritmi simmetrici**: utilizzano la stessa chiave, oppure è semplice derivare una dall'altra;

- **algoritmi asimmetrici o chiave pubblica**: chiavi diverse, non derivabili tra loro; la pubblicazione della *public key* non compromette la *private key*.



Ci sono diverse classificazioni di sicurezza:

- *sicurezza incondizionale*: sistema sicuro anche se l'avversario ha un'infinita potenza di calcolo poiché il testo cifrato non ha abbastanza informazioni per risalire all'originale; la sicurezza si misura con la *information theory*.
- *sicurezza condizionale*: il sistema può essere attaccato ma sarebbe richiesta una potenza di calcolo irreale; la sicurezza è misurata usando la *complexity theory*.

Cryptoanalysis: scienza che si occupa di recuperare un testo in chiaro partendo dal testo cifrato, senza chiave:

- *brute force attacks*: è sempre possibile, provando ogni chiave. È molto costoso sulla dimensione della chiave.
- *cryptoanalytic attacks*: l'attaccante inizialmente conosce qualcosa (public key, plain texts . . .); un oracolo (informazioni che ottiene durante un attacco).

Simmetric Key Encryption

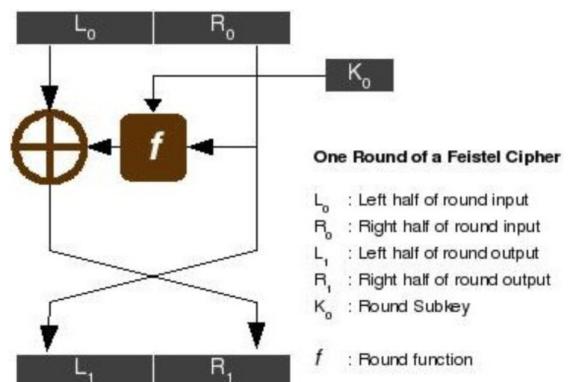
Uno schema crittografico è a chiave simmetrica se per ogni coppia di chiavi (e, d) , esse sono uguali oppure è facile derivare una dall'altra. Conosciuti anche come: *single-key*, *one-key*, *shared-key*, *conventional encryption*.

Si può anche utilizzare il *cifrario di Cesare*, oppure sostituire le lettere con la posizione nell'alfabeto.

Composite Ciphers: cifrare basandosi solo su sostituzioni o trasposizioni non è sicuro. È meglio una sostituzione seguita da una trasposizione. Difficile da fare a mano: invenzione delle macchine di cifratura (*confusion and diffusion*). La fase di *diffusion* nasconde la struttura del testo nel messaggio cifrato, la fase di *confusion* rende molto complessa la relazione tra testo cifrato e chiave.

Un esempio pratico è la **Feisal Cipher Structure**.

Così come l'algoritmo **DES** (*Data Encryption Standard*).

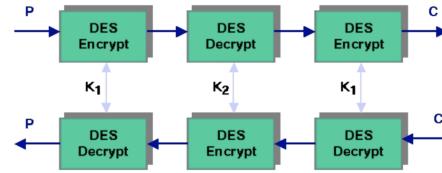
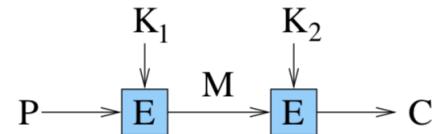


L'algoritmo **DES** si è rivelato essere poco sicuro perché la chiave usa solamente 56 bit per la cifratura dei blocchi di 64 bit. Raddoppiando il numero di chiavi (**Double DES**) non si ottiene un algoritmo con chiavi di 112 bit.

Resta esposto all'attacco *Meet-in-the-Middle*.

Tripliando le fasi crittografiche (**Triple DES**), usando le chiavi K_1 e K_2 , si ottiene un algoritmo più sicuro poiché richiede un *brute-force attack* di complessità 112 bit.

Attualmente non esistono attacchi pratici all'algoritmo ed è compatibile con lo standard *DES* ($K_1 = K_2$).



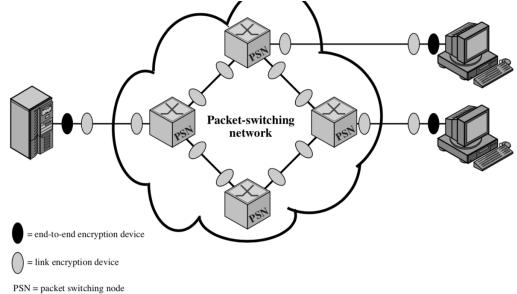
Come successore di *DES* è stato proposto **AES** (*Advanced Encryption Standard*) che permette di criptare blocchi di 128, 192, 256 bit (non solo 64). Non usa la struttura di *Feistal* ma processa l'intero blocco in parallelo.

Quando un messaggio supera la dimensione del blocco, dividerlo non è la soluzione migliore perché si possono perdere informazioni (stessi blocchi cifrati mappati sugli stessi blocchi in chiaro) e limitare l'integrità (la fase di decriptazione non indica se un blocco è stato cambiato, eliminato o duplicato).

È meglio fare lo XOR tra un blocco in chiaro e il precedente cifrato, così blocchi identici sono mappati in blocchi diversi cifrati. Se ci sono problemi al blocco C_j ma non al blocco C_{j+1} , il blocco C_{j+2} viene correttamente decriptato.

Placement of Encryption

Link Encryption: la cifratura viene effettuata indipendentemente in ogni collegamento; il traffico deve essere decifrato e criptato ad ogni link (servono molti device, ma chiavi pari). Viene svolta al livello 1 o 2 dello stack OSI.



End-to-End Encryption: la cifratura avviene solo tra la sorgente e la destinazione; richiede chiavi condivise. Viene svolta ai livelli 3, 4, 6, 7 dello stack OSI (più sicuro).

Nella crittografia *end-to-end*, gli headers vengono lasciati in chiaro per essere letti dalla rete e diretti alla destinazione. Idealmente, vorremmo poter criptare anche quelle informazioni per evitare che il monitoring della rete.

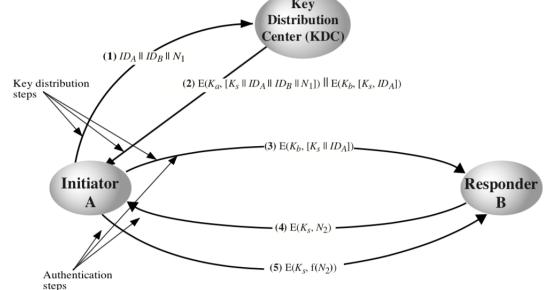
Key Distribution

Gli schemi simmetrici prevedono che entrambe le parti si scambino delle chiavi segrete comuni, in modo sicuro.

- A può selezionare la chiave e spedirla fisicamente a B ;
- una terza parte può scegliere la chiave e spedirla ad A e B ;
- se A e B hanno comunicato in precedenza, possono utilizzare la chiave precedente per criptarne una nuova;
- se A e B hanno una comunicazione sicura con una terza parte C , allora C può trasmettere la chiave tra A e B .

Soltanamente viene utilizzata una gerarchia di chiavi:

- **Session key:** usata per la cifratura dei dati tra due utenti per una sessione e poi viene scartata;
- **Master key:** usata per criptare le *session key*; condivisa tra gli utenti e il *key distribution center*.



Le chiavi di sessione distribuite dal *KDC* hanno un tempo di vita limitato per garantire maggiore sicurezza. Inoltre fa uso di un sistema di distribuzione automatico tra utenti (decentralizzato). Deve controllare l'utilizzo delle chiavi.

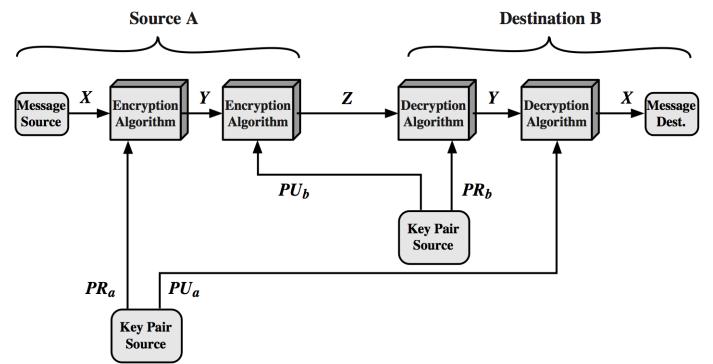
Algoritmi di Crittografia a Chiave Pubblica

La *crittografia a chiave pubblica* nasce nel 1975 per risolvere due problemi: la distribuzione delle chiavi e la firma. Considerando la coppia (E_e, D_d) , anche se E_e è noto, è impossibile risalire da $c = E_e(m)$ a m ; serve D_d . Per questo motivo si può affermare che la *public key* può essere un'informazione pubblica (a differenza della *private key*).

Cifratura di un messaggio M : $C = E(PU_b, M)$

Decifratura di un messaggio C : $M = D(PR_b, C) = D(PR_b, E(PU_b, M))$

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



Un algoritmo di crittografia a chiave pubblica funziona come segue:

- si utilizza lo stesso algoritmi per le fasi di *encryption* e *decryption* con una coppia di chiavi (una per fase);
- mittente e destinatario devono avere entrambi i tipi di chiave (non uno solo).

Un algoritmo di crittografia a chiave pubblica, per la sicurezza ha bisogno delle seguenti proprietà:

- una delle due chiavi deve rimanere segreta;
- è praticamente impossibile decifrare un messaggio senza informazioni aggiuntive;
- conoscere l'algoritmo, una sola chiave e dei modelli del testo cifrato non è sufficiente per risalire all'altra chiave.

Un destinatario B genererà le chiavi PU_b e PR_b , il mittente A conoscerà PU_b e M per generare $C = E(PU_b, M)$; il destinatario B potrà decriptare C con PR_b . L'avversario, dati PU_b e C , non riesce a ricostruire M .

Questo si ha perché l'algoritmo si basa su una **one-way function**: data $f : X \rightarrow Y$, f è facile da calcolare per ogni $x \in X$, ma f^{-1} è difficile da trovare.

Si basa inoltre sul concetto di **trapdoor function**: una funzione $f_k : X \rightarrow Y$ che data una informazione extra k , è semplice trovare per $y \in Im(f)$, $x \in X$ dove $f_k(x) = y$

Public-key Cryptoanalysis

- *Brute-force attacks*: la contromisura consiste nell'utilizzo di chiavi grandi; è però necessario un tradeoff per non aumentare troppo la complessità delle fasi di encryption/decryption.
- *Computing private key from public key*: non ci sono prove che questo attacco sia irrealizzabile.
- *Probable-message attack*: supponendo di avere un messaggio M di pochi bit (es. 56bit) criptato con la PU_a , l'attaccante può calcolare tutti i possibili $Y_i = E(PU_a, X_i)$ per tutti i possibili messaggi $X_i, i = 1 \dots 2^{56}$! La soluzione consiste nell'aggiungere bit random a M .

Algoritmo RSA

Algoritmo pubblicato nel 1976 e nominato nel 1978 dagli inventori Rivest, Shamir, Adleman.

La sua sicurezza è data dalla difficoltà di fattorizzare grandi numeri. Le chiavi sono funzioni di coppie di grandi numeri primi (oltre 100 cifre). È l'algoritmo a chiave pubblica più popolare, utilizzato in molte applicazioni (PGP, SSL...). Risolve il problema della gestione delle chiavi ma ha alcuni **svantaggi**: lentezza, ha bisogno di chiavi grandi ed è vulnerabile ad alcuni attacchi (*plaintext e timing attacks*).

Number Theory

- *Prime Factorization*: scrittura di un numero come prodotto di numeri primi.
- *Relative Prime Numbers & MCD*: due numeri sono relativamente primi se non hanno divisori comuni; tra due numeri si può trovare anche il *massimo comune divisore*.
- *Modular Arithmetics*: $\forall a, n. \exists q, r. (a = q * n + r)$ con $0 \leq r < n < a$, il resto $r = a \bmod n$.
 $a, b \in \mathbb{Z}$ sono *congruenti modulo n* se $a \bmod n = b \bmod n$ e si scrive con $a =_n b$.
- *Euler Totient Function*: facendo il modulo n si ha un insieme completo di resti di cui si tengono solamente i numeri primi (*reduced set of residues*), il cui numero di elementi è chiamato **Euler Totient Function** $\phi(n)$.

Teorema 1 (Fermat's little theorem). *Per a e n primi tra loro ed n primo, $a^{n-1} =_n 1$*

Considerato $a \bmod n$:

Residui: $0, \dots, n - 1$

Set di residui ridotto: *residui che sono primi tra loro con n*

Il numero di elementi nel **set di residui ridotto** è chiamata **funzione di totient di Eulero**: $\phi(n)$

Proprietà:

- $\phi(1) = 1$
- $\phi(p) = p - 1$ se p è primo
- $\phi(p * q) = \phi(p) * \phi(q) = (p - 1) * (q - 1)$ se p e q sono primi e $p \neq q$

Il Teorema 1 può essere riscritto come:

Teorema 2 (Teorema di Eulero). $a^{\phi(n)} =_n 1$ per tutti gli a, n tali che $MCD(a, n) = 1$

Funzionamento

Cifratura: $C = M^e \bmod n$

Decifratura: $M = C^d \bmod n = (M^e)^d \bmod n = M^{ed} \bmod n$

Chiave pubblica: $PU = e, n$

Chiave privata: $PR = d, n$

Generazione chiavi:

- Generare due numeri primi p e q distinti e molto grandi;
- Computare $n = pq$ e $\phi = (p - 1)(q - 1)$
- Selezionare e tc. $1 < e < \phi$ relativamente primo a ϕ ;
- Computare $d = e^{-1} \bmod \phi$;
- Chiave pubblica = (e, n)
Chiave privata = (d, n)

Un eventuale progresso futuro nella *teoria dei numeri* potrebbe rendere **RSA** insicuro poiché si basa sulla difficoltà di effettuare certe computazioni in un tempo ragionevole.

Criptazione:

- Dividere il messaggio in blocchi $M_1, M_2 \dots$ con $M_i < n$
- Computare $C_i = M_i^e \text{ mod } n$

Decriptazione:

- Computare $M_i = C_i^d \text{ mod } n = (M_i^e)^d \text{ mod } n = M_i^{ed} \text{ mod } n$

Condivisione Segreta delle Chiavi con RSA

Essendo la crittografia a chiave pubblica molto onerosa dal punto di vista computazionale, si ricorre alla crittografia simmetrica, in cui lo scambio iniziale delle chiavi avviene utilizzando la crittografia asimmetrica.

Criptazione (con chiave pubblica (e, n)):

- Scegliere randomicamente k
- $C = (k^e \text{ mod } n, E_k(m))$

Decriptazione (con chiave privata (d, n)):

- Splittare il messaggio C in (c_1, c_2)
- $k = c_1^d \text{ mod } n$
 $m = D_k(c_2)$

Problema: se la private-key (d, n) è compromessa, k può essere recuperata da un intruso che registra la conversazione.

Algoritmo di Diffie-Hellman

Fondamenti matematici

Una **radice primitiva** s di un numero primo p è un numero le cui potenze generano $1, \dots, p - 1$. Così $s \text{ mod } p, \dots, s^{p-1} \text{ mod } p, \dots$ sono permutazioni distinte da 1 a $p - 1$. Quindi

$$\forall b \in \mathbb{Z}. \exists i \in \{0, \dots, p - 1\}. b = s^i \text{ mod } p$$

Dato $b \in \mathbb{Z}$, l'esponente i è il **logaritmo discreto** di b per la base s , mod p . Computare **logaritmi discreti** sembra essere computazionalmente impossibile.

Funzionamento

L'obiettivo è lo scambio di una chiave condivisa tra due attori:

1. Gli attori A e B si scambiano un *numero primo* q e una *radice primitiva* α , entrambi pubblicamente visibili;
2. A e B generano numeri casuali X_A e X_B , rispettivamente minori di q ;
3. A computa $Y_A = \alpha^{X_A} \text{ mod } q$, B computa $Y_B = \alpha^{X_B} \text{ mod } q$;
4. A e B si scambiano i risultati;
5. A computa $K_A = Y_B^{X_A} \text{ mod } q$, B computa $K_B = Y_A^{X_B} \text{ mod } q$. Le **chiavi** sono **uguali**, infatti:

$$K_A = Y_B^{X_A} \text{ mod } q = (\alpha^{X_B})^{X_A} \text{ mod } q = (\alpha^{X_A})^{X_B} \text{ mod } q$$

Punti di forza:

- La chiave condivisa è creata dal nulla;
- La chiave condivisa ottenuta è almeno potente quanto la *half-key*;
- La chiave condivisa non è mai trasmessa, se non in forma criptata (si scambiano le *half-key*);
- **Perfect Forward Secrecy (PFS):** una chiave di sessione derivata da un insieme di chiavi private e pubbliche non sarà mai compromessa se una di esse viene violata. Verranno violati solamente i dati criptati con le chiavi compromesse.

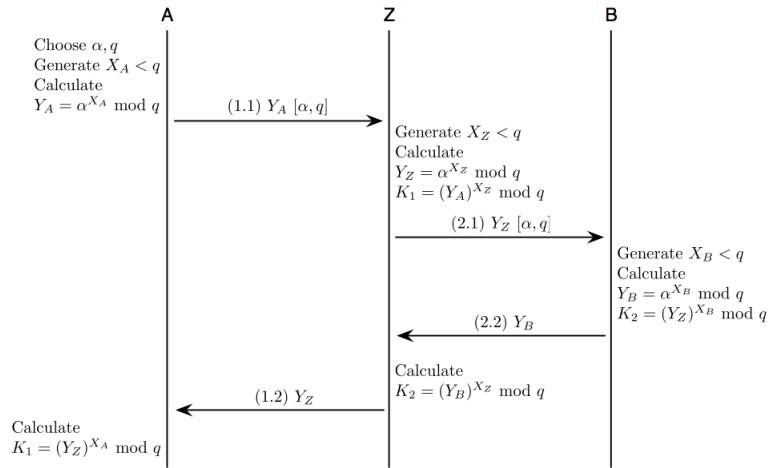
Se qualcuno dovesse scoprire le chiavi private dopo aver registrato l'intera conversazione, non sarebbe comunque in grado di decriptare nulla (tranne i dati esplicitamente criptati con quella chiave privata).

Questo algoritmo gode comunque della proprietà di **scalabilità** poiché può essere esteso a 3 o più utenti.

Debolezze: vulnerabile a **MitM attack** nella variante non autenticata.

- A trasmette Y_A a B
- Z (l'attaccante) intercetta Y_A e trasmette Y_Z a B. Calcola anche $K_1 = (Y_A)^{X_Z} \bmod q$
- B riceve Y_Z e calcola $K_2 = (Y_Z)^{X_B} \bmod q$
- B trasmette Y_B ad A
- Z intercetta Y_B e trasmette Y_Z ad A. Calcola anche $K_2 = (Y_B)^{X_Z} \bmod q$
- A riceve Y_Z e calcola $K_1 = (Y_Z)^{X_A} \bmod q$

In questo caso A pensa di condividere una secret-key ma sta condividendo K_1 con Z, mentre B condivide K_2 con Z. Ciò è dovuto alla **mancanza di autenticazione** tra A e B (non c'è nessuna violazione essendo su canale pubblico).



ElGamal (Variante di Diffie-Hellman)

Funzionamento

Simile a **Diffie-Hellman**: al passo 2 viene già inviato il messaggio criptato

1. B calcola $Y_B = \alpha^{X_B} \bmod q$ e poi $B \rightarrow A : Y_B$
2. A calcola $Y_A = \alpha^{X_A}$ e anche $K = Y_B^{X_A} \bmod q$; infine $A \rightarrow B : (E(M, K), Y_A)$
3. B computa $K = Y_A^{X_B} \bmod q$ e usa K per decriptare $E(M, K)$

Massey-Omura

Funzionamento

Criptazione senza chiavi condivise, si basa sul problema del logaritmo discreto, condividendo un numero primo p ;

- $u \in \{A, B\}$ sceglie $e_u, d_u \in \mathbb{Z}$ in modo privato, tale che: $e_u d_u \bmod (p - 1) = 1$
Quindi $(p - 1)$ divide $(e_u d_u - 1)$. Ricordando che se $a/b \mapsto a = q * b + r$, esiste un k dove $e_u d_u = k(p - 1) + 1$.
Dal teorema di Eulero, $\forall m \in \{1, \dots, p - 1\}$

$$m^{e_u d_u} \bmod p = m^{k(p-1)} m \bmod p = m \bmod p = m$$

- Schema:

1. $A \rightarrow B : m^{e_A} \bmod p$
2. $B \rightarrow A : m^{e_A e_B} \bmod p$
3. $A \rightarrow B : m^{e_A e_B d_A} \bmod p (= m^{e_B})$
4. $A \rightarrow B : m^{e_A e_B d_A d_B} \bmod p (= m)$

Anche in questo caso A NON è certa di scambiare messaggi con B: ancora possibile **MitM attack**.

Integrità dei Messaggi e Hash Crittografici

Hash

Per integrità del dato si intende la proprietà che esso non sia in qualche modo alterato da terze parti.

Si vuole creare un'impronta digitale del dato, cioè una **funzione di hashing** $h(x)$, che ha le seguenti proprietà:

1. *Compressione*: h mappa un'input x di lunghezza arbitraria, in un output $h(x)$ di lunghezza fissata n ;
2. Calcolabile in tempo polinomiale.

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

1. è *non invertibile* (detta anche proprietà di *pre-image resistant*)
2. solitamente anche:
 - *2-nd preimage resistant*: impossibile trovare un secondo input che ha lo stesso output del primo, cioè dato x , è difficile trovare $x' \neq x$. $h(x) = h(x')$
 - *Collision resistance*: difficile trovare due input distinti x, x' . $h(x) = h(x')$

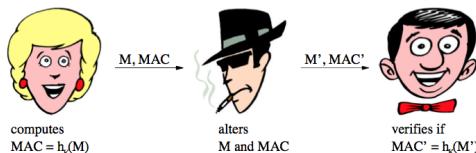
Per costruire una *funzione di hash crittografica* si può usare la tecnica della **block chain**: l'output dell'encoding di un blocco viene dato in input al seguente per essere utilizzato nell'encoding dello stesso (e così via per gli altri blocchi).

MAC

L'obiettivo è sempre quello di mantenere l'**authentication integrity** per verificare che nessuno alteri la conversazione. Il **Message Authentication Code (MAC)** e la firma digitale sono le due tecniche principali per l'autenticazione.

Un algoritmo MAC è una famiglia di funzioni di hash h_k parametrizzate da una chiave segreta k condivisa. h_k deve essere *computazionalmente resistente*: date 0 o più coppie MAC $(x_i, h_k(x_i))$ non è fattibile computare $(x, h_k(x))$ per ogni nuovo input $x \neq x_i$ (è impossibile calcolare l'hash se non si conosce la chiave segreta k).

Non c'è però il controllo sui message replay (la firma può essere scaduta), dunque non assicura *freshness*.



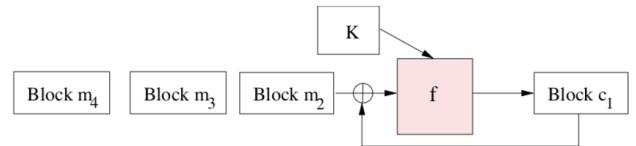
Costruire un algoritmo MAC

Una realizzazione si basa sulla *cipher-block chaining*.

$$c_1 = E_K(m_1 + 0)$$

$$c_i = E_K(c_{i-1} + m_i)$$

E è un blocco cifrato (es. DES), c_n è il MAC.



Firma digitale

Problema della prova dell'origine dei dati: si può essere sicuri che il messaggio provenga da una persona specifica (**proprietà di non ripudio**) utilizzando una chiave condivisa?

La firma digitale è fondamentale per l'autenticazione e il *non ripudio* dei messaggi.

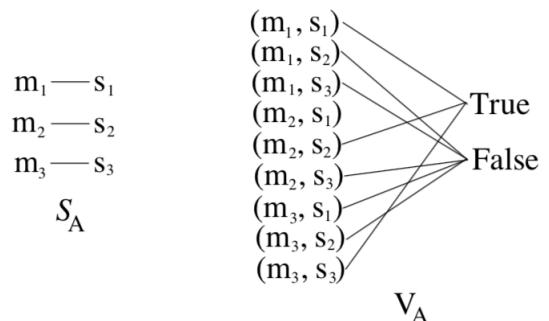
Sono tuttavia possibili i *replay attack* poiché le firme possono essere scadute.

Nomenclatura

- M è l'insieme dei messaggi che possono essere firmati;
- S è l'insieme delle firme, cioè stringhe di n bit;
- $S_A : M \rightarrow S$ è la trasformazione corrispondente alla firma per A , tenuta segreta da A ;
- $V_A : M \times S \rightarrow \{true, false\}$ è la funzione di verifica per A ed è nota pubblicamente (prende un messaggio ed una firma: ritorna *true* se c'è corrispondenza);
- S_A e V_A forniscono uno schema di **firma digitale per A**

Costruzione di uno schema di firma

- *Signing procedure*: A crea una firma per $m \in \mathcal{M}$ creando $s = S_A(m)$ e trasmettendo la coppia (m, s) .
- *Verification procedure*: B verifica la firma di A su (m, s) calcolando $u = V_A(m, s)$ e la accetta se ritorna *true*.
- *Secrecy requires*: è difficile per un'entità diversa da A trovare, per ogni $m \in \mathcal{M}$ un $s \in \mathcal{S}$, tale che $V_A(m, s) = true$.



L'implementazione della firma digitale si può basare sui sistemi (reversibili) di crittografia a chiave pubblica.

Sia $E_e : \mathcal{M} \rightarrow \mathcal{C}$ una trasformazione a chiave pubblica e supponiamo che $\mathcal{M} = \mathcal{C}$. Se D_d è la funzione di decriptazione corrispondente a E_e , allora entrambe sono permutazioni:

$$D_d(E_e(m)) = E_e(D_d(m)) = M \quad \text{per ogni } m \in \mathcal{M}$$

Questo schema di crittografia a chiave pubblica è chiamato *reversibile*.

La costruzione dello schema per la firma digitale avviene come segue:

- siano \mathcal{M} e \mathcal{C} un messaggio e uno spazio di firma con $\mathcal{M} = \mathcal{C}$;
- sia (e, d) una coppia di chiavi per lo schema di crittografia a chiave pubblica;
- definiamo che la funzione di firma S_A sia D_d , cioè $s = D_d(m)$;
- definiamo $V_A(m, s) = true$ se $E_e(s) = m$, altrimenti è *false*.

Lo schema ammette il **forgery attack**: l'attaccante B sceglie randomicamente $s \in \mathcal{S}$ e calcola $m = E_e(s)$; poiché $\mathcal{S} = \mathcal{M}$, sottomette il messaggio con la firma e la verifica ritorna *true* anche se A non ha firmato m !

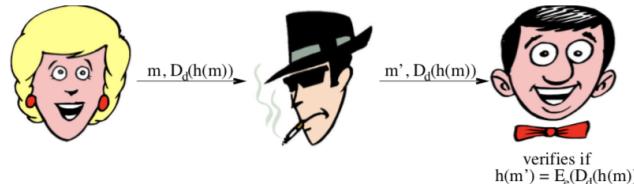
Soluzione: sia $\mathcal{M}' \subset \mathcal{M}$ un sottoinsieme di messaggi firmabili; si ridefinisce $V_A : \mathcal{S} \rightarrow \{true, false\}$ come:

$$V_A(s) = true \quad \text{se } E_e(s) \in \mathcal{M}', \text{ altrimenti è } false$$

I messaggi sono recuperabili da $M = E_e(s)$. È sicuro quando \mathcal{M}' è un sottoinsieme sufficientemente piccolo di \mathcal{M} .

La **falsificazione (forgery)** è **prevenuta** firmando i messaggi con una struttura fissa:

- il nome del messaggio è il suo mittente, oppure (solitamente);
- l'hash crittografico firmato è spedito con il messaggio (la coppia può essere criptata per la confidenzialità).



Dunque le crittografie simmetrica/asimmetrica assicurano la confidenzialità distribuendo propriamente le chiavi. La crittografia asimmetrica semplifica la distribuzione delle chiavi (ma ha bisogno di un canale autenticato). È importante implementare l'autenticazione dei messaggi, una volta che le chiavi sono state distribuite.

Public-Key Infrastructures

È importante che esista un *key management* per: distribuzione di chiavi crittografiche, creare un legame tra un'identità ed una chiave, generare, mantenere e revocare le chiavi (possono essere state rubato o scadute).

Un **PKI** è un'infrastruttura che permette agli attori di riconoscere l'appartenenza di una *public-key a qualcuno*. Per fare parte del **PKI**, un utente deve:

- generare la sua coppia di chiavi privata/pubblica;
- inviare la sua chiave pubblica ad un **Certification Authority (CA)** affidabile per tutti;
- il **CA** verifica che l'utente sia colui che dice di essere e infine firma un *certificato digitale* che assicura l'appartenenza di quella chiave pubblica al determinato utente.
- tutti gli altri utenti adesso possono verificare il certificato per ottenere la chiave pubblica di quell'utente.

Le **componenti di un PKI** sono le seguenti:

- **Certification Authority (CA)**: crea i certificati e li pubblica nella *directory*; mantiene anche il *Certificate Revocation List (CRL)* nella *directory*.
- **Certificate Revocation List (CRL)**: interpellato per sapere se i certificati sono ancora validi. La **CA** può anche revocare i certificati per motivi diversi. Si occupa del *Key Recovery* in caso di smarrimento della chiave privata, di firma o di non ripudiazione (solitamente viene invalidata e creata una nuova).
- **Directory**: rende disponibili i certificati degli utenti (per identificarli univocamente) e il *CRL*;
- **Registration Authority (RA)**: si occupa del processo di registrazione degli utenti, assicurandone l'identità.
- **Clients**: gli utenti che si registrano e autenticano.

Le **PKI** possono essere **Open** (usate per compagnie e comunità) oppure **Closed** (limitate ad un gruppo di utenti).

Certificati

Un **certificato** è un token che lega un'identità ad una chiave.

L'utente *C* firma (con la sua chiave privata) un messaggio contenente la rappresentazione dell'identità di *Alice*, la corrispondente chiave pubblica (PU_{Alice}) e il *timestamp T*:

$$C_{Alice} = M || E(PR_C, H(M)) \text{ dove } M = \langle PU_{Alice}, Alice, T \rangle$$

B può verificare il certificato per ottenere la chiave pubblica di *Alice* e accettarne la validità. Tuttavia *B* può conoscere la chiave pubblica di *C* per validare il certificato.

Il problema è un altro adesso: come può essere convalidato il certificato dell'emittente?

- costruire un albero gerarchico, con la chiave pubblica della radice conosciuta fuori dalla banda;
- oppure permettere una arrangiamento arbitrario affidandosi sulla conoscenza di ciascuno riguardo i certificatori.

X.509

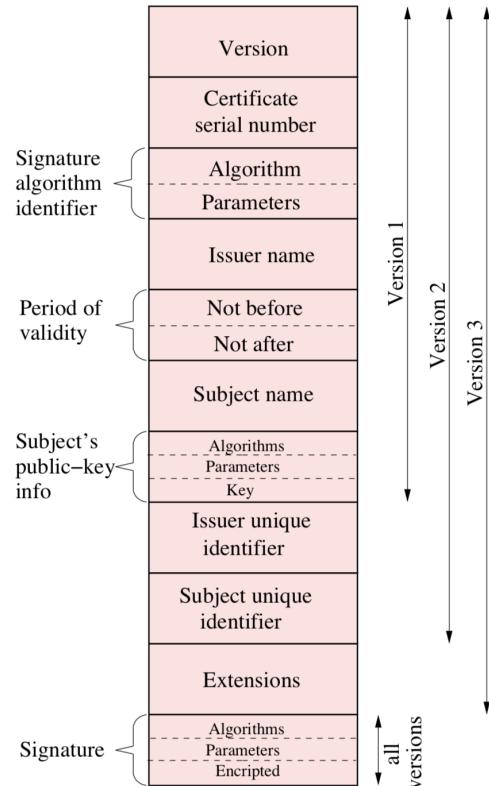
Per illustrare questi approcci, i certificati e le certificazioni, si prende come esempio il **X.509**.

X.509 è uno standard per l'utilizzo dei servizi di autenticazione. La struttura del certificato ed i protocolli di autenticazione in esso definiti sono utilizzati in molti contesti (*IPSEC, SSL/TLS, SET e S/MIME*).

Si basa sulla crittografia a chiave pubblica (raccomanda *RSA*), funzioni di hash e firme digitali.

Il cuore dello schema **X.509** è il certificato della chiave pubblica associato ad ogni utente. Viene creato dal *CA* e messo nella *directory* che garantisce un accesso semplice agli utenti per ottenere i certificati.

- *Serial Number*: deve essere univoco in coppia al numero dell'emittente;
- *Signature Algorithm Identifier*: identifica l'algoritmo, e ogni parametro utilizzato per firmare il certificato;
- *Issuer Name*: il nome del *CA* che ha creato e firmato il certificato;
- *Period of Validity*: scadenza del certificato;
- *Subject Name*: nome dell'utente al quale il certificato si riferisce;
- *Subject Public-Key Info*: identifica l'algoritmo, i parametri e la chiave pubblica dell'utente;
- *Signature*: contiene l'hash code degli altri campi, crittati con la chiave privata del *CA*.



L'utente riceve PU_{CA} , decodifica il campo *Signature*, computa l'hash del certificato e ne verifica la validità.

Trust Models

Esistono diversi **Trust Models** che dettano come gli utenti devono stabilire la validità di un certificato: *Direct Trust*, *Hierarchical Trust*, *Web of Trust*.

Direct Trust: se tutti gli utenti sono iscritti alla stessa *CA*, c'è una fiducia comune verso di essa. Tutti i certificati si trovano nella stessa *directory* per l'accesso da parte degli utenti che possono anche trasmetterseli (come in *PGP*).

Hierarchical Trust: per grandi community è più pratico avere più *CA*, creando un **Trust Tree** che si estende da un numero di *certificati radice*. Questi certificati possono certificare i certificati stessi oppure altri certificati che certificano altri certificati lungo la catena. La validità del *certificato foglia* è verificata risalendo dal suo certificatore, ad altri certificatori, fino a quando non viene trovato un certificato radice attendibile.

Cross Certification: se *A* (con certificato nel *CA* X_1) vuole il certificato di *B* (in X_2), può fare affidamento sul fatto che i due *CA* si siano scambiati le chiavi pubbliche. Dunque *A* può ottenere il certificato di *B* in una *catena di certificati*. Lo schema X.509 prevede appunto che i vari *CA* siano disposti secondo una *gerarchia*.

Web of Trust: comprende *direct* e *hierarchical trust*, aggiungendo l'idea che il trust sta nell'occhio dell'osservatore (visione del mondo reale) e che più informazioni sono migliori. Un certificato può essere *trusted* direttamente oppure tramite una catena fino a risalire alla radice o ad un gruppo di *introducers* (catene di fiducia).

Web of Trust: PGP

I certificati **PGP** differiscono da **X.509** in alcuni punti importanti:

- una chiave *PGP* può avere molteplici firme (anche del proprietario) per aumentare la fiducia in quel certificato;
- ogni firma è una nozione di fiducia e le firme possono avere diversi livelli di fiducia (gli utenti agiscono in base a questo livello). Si tratta dunque di *Reputation System* poiché non vi è l'autorità centrale.

Nell'ambiente *PGP*, ogni utente agisce come un *CA* firmando gli altri certificati. Firmando l'utente ne diventa un *introducer*. Per definire inoltre se un certificato è valido è necessario conoscere uno dei firmatari oppure ricostruire una catena. Si stabilisce un *web of trust*.

In *PGP* i certificati indicano dunque un livello di fiducia che però può avere diversi significati per differenti firmatari.

Network Security and Internet Protocols

I protocolli sono essenziali per sviluppare servizi web poiché il *mondo è distribuito*. Un protocollo è un insieme di regole (convenzioni) che determinano l'ordine e le *regole* per lo scambio di messaggi tra due o più attori; in breve è un *algoritmo distribuito* che si occupa della comunicazione.

Un **protocollo di sicurezza** (oppure **crittografico**) usa meccanismi di crittografia per raggiungere obiettivi quali: autenticazione degli utenti, scambio di chiavi, integrità ...

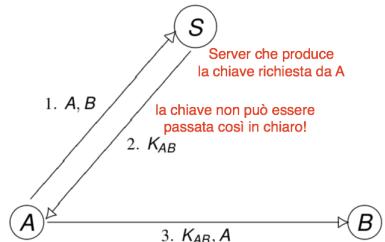
Costruzione di un Protocollo per lo Sambio di Chiavi

Per progettare un buon protocollo, bisogna stabilire l'architettura per le comunicazioni: si hanno due utenti (mittente A e destinatario B) che vogliono creare una chiave di sessione in modo sicuro e protetto. Si dovranno appoggiare ad un server S assunto come affidabile. Il server dovrà produrre una chiave di sessione segreta e autenticata da inviare (sempre in modo sicuro) ai due attori.

Scopo finale del protocollo: A e B devono conoscere la chiave di sessione (e nessun altro tranne S !) e sapere che è stata appena generata (proprietà di *freshness* contro i *replay attack*).

Tentativo 1: protocollo che si basa su 3 messaggi.

- **Problemi:** La chiave viene passata in chiaro. Possiamo dichiarare che: l'avversario può intercettare tutti i messaggi.

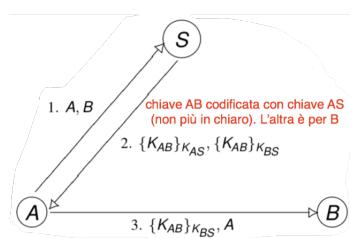


Tentativo 2: il server S manda le chiavi criptate per evitare che l'avversario possa ottenerle in chiaro.

- **Problemi:** l'avversario è in grado di alterare tutti i messaggi in transito usando qualsiasi informazione disponibile ed eseguirne il forwarding, forgiare e inserire nuovi messaggi (ha il controllo della rete).

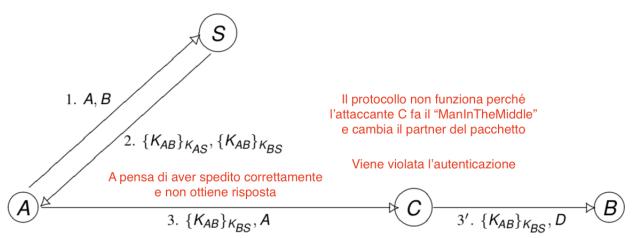
Si possono verificare due casi di attacco in particolare:

- B crede di condividere la chiave con A , ma in realtà la sta condividendo con D !

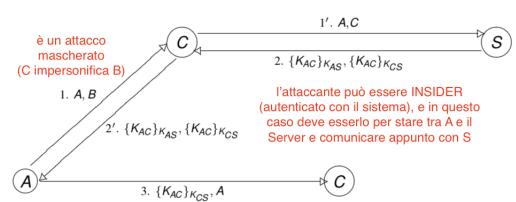


- C altera i messaggi che A manda ad S , così S critpta i messaggi con la chiave di C . D'ora in poi A pensa di spedire a B ma in realtà sta comunicando con C . È un **masquerading attack** perché C impersonifica B a causa della mancanza di **autenticazione**.

In questo caso C può essere un legittimo partecipante della rete (*insider*) oppure un *outsider*, o addirittura una combinazione di entrambi.



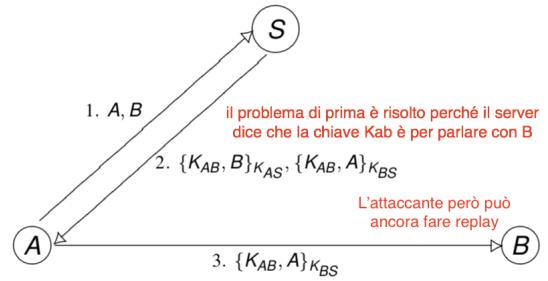
L'avversario può essere uno dei due attori (*insider*) o una parte esterna (*outsider*) o una combinazione delle due.



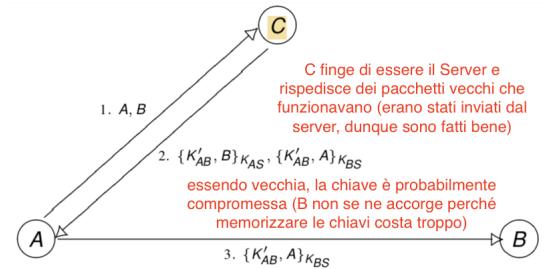
Tentativo 3: il server S manda le chiavi criptate indicando anche i nomi degli attori.

Ora l'avversario non può origliare o alterare la conversazione tra le parti!

- **Problemi:** l'avversario può ottenere il valore della chiave di sessione K_{AB} usata in un'esecuzione precedente del protocollo.



- C intercetta i messaggi da A ad S
- **Replay attack:** K'_{AB} è una chiave vecchia usata da A e B in una sessione precedente. Viene meno la **freshness** della chiave. Per ovviare al problema si usano i **nonce**.



Tentativo 4: noto come **Needham e Schroeder** (1978).

- È attualmente uno dei protocolli più conosciuti ma è vulnerabile all'attacco fatto da *Denning e Sacco*. Si basa sui **nonce** (*number used only once*), cioè un valore generato randomicamente da uno degli utenti per dimostrare che il messaggio è nuovo (lo riceverà anche nella risposta).
- **Problemi:** se il protocollo abortisce, A non è sicuro che B abbia ricevuto o meno la chiave (è necessaria la **key confirmation**).

- C , mascherato come A può inviare una vecchia chiave a B ; nel frattempo A , non ricevendo risposte da B , abortisce il protocollo. Mentre B continua la conversazione con C utilizzando una chiave corrotta (senza sapere che la chiave è corrotta e che dall'altra parte non vi è realmente A).

Tentativo 5: **Kerberos**.

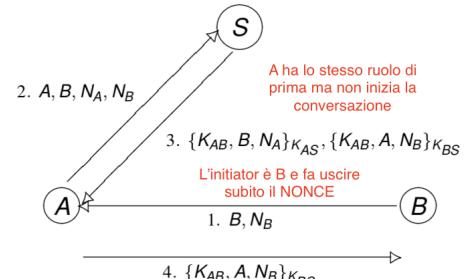
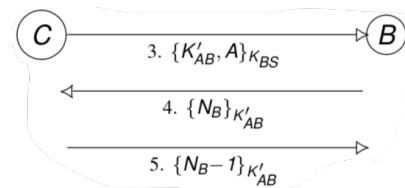
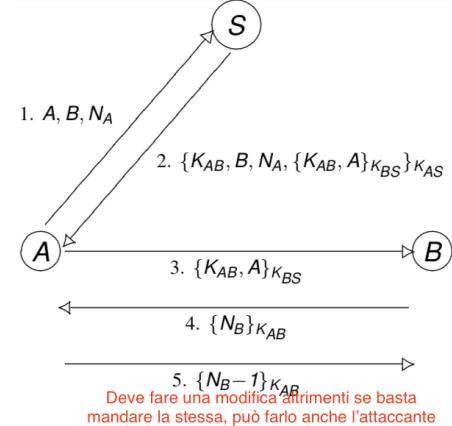
- L'**initiator** è B che manda il suo **nonce** e identità ad A ; quest'ultimo aggiunge il suo **nonce** e lo manda al server per ricevere la chiave.

Infine A manda la chiave, la sua identità e il **nonce** di B al mittente (criptato con la sua chiave).

La proprietà di **key confirmation** è raggiunta sapendo che il messaggio 4 può essere creato solamente conoscendo K_{AB} .

Alla fine del protocollo conclusivo, nessuno dei due attori ha la certezza che l'altro abbia ricevuto K_{AB} .

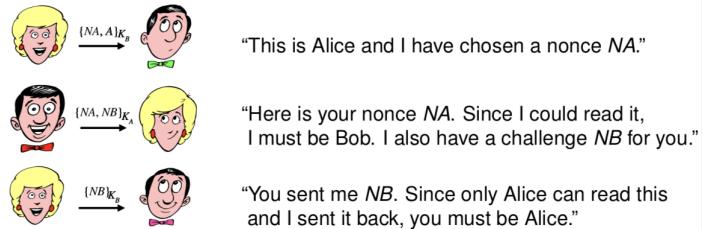
Questo protocollo evita tutti gli attacchi visti in precedenza. È comunque azzardato affermare che esso sia sicuro. La sicurezza di un protocollo si può definire in relazione agli obiettivi da raggiungere (da formalizzare con esso).



Il Protocollo di Needham-Schroeder a Chiave Pubblica (NSPK)

Il protocollo NSPK (nato nel 1970) ha i seguenti obiettivi:

- *mutua autenticazione* legando messaggi e mittente;
- Assicurare che i messaggi siano recenti (*timeliness*);
- garantire la *secrecy* di alcuni elementi (chiavi).



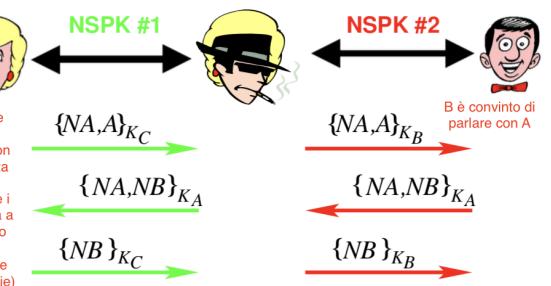
MiTM di tipo masquerading attack

Ciò è dovuto all'assenza del **binding** al passo 2:

$$B \rightarrow A : \{NA, NB, B\}.$$

È molto importante legare al *nonce* il nome del mittente, altrimenti il protocollo non è sicuro.

La versione migliorata è il protocollo NSL.



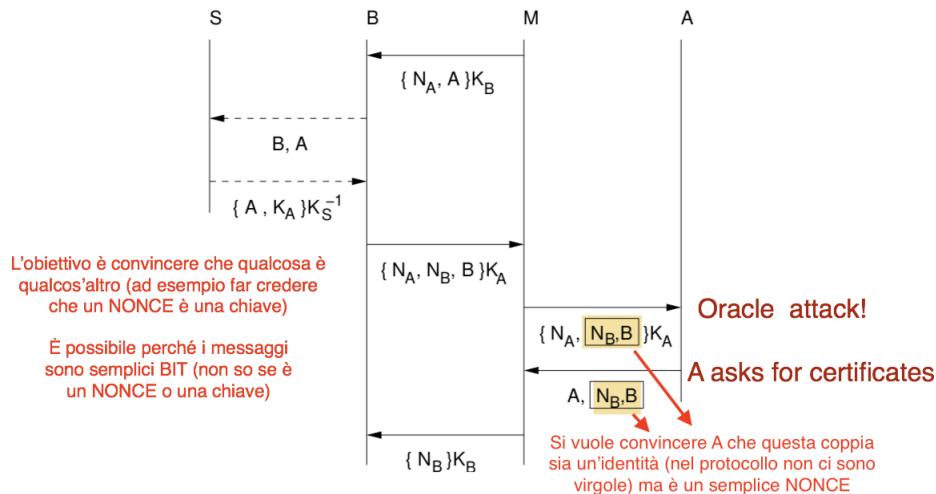
Il Protocollo NSL (Needham-Schroeder-Lowe)

Se consideriamo **autenticazione** e **segretezza**, questo protocollo è sicuro.

Considerando invece l'**aliveness** degli attori (se *B* volesse solamente sapere se *A* è *alive*), quello presentato in figura può essere considerato un attacco!



Type-Flaw & Oracle Attack



Un'altra soluzione consiste nello scambio dei *nonce* firmati con la chiave privata del mittente.

- M1.** $A \rightarrow B : \{[N_A, A]_{K_A^{-1}}\}_{K_B}$
- M2.** $B \rightarrow A : \{N_A, [N_B]_{K_B^{-1}}\}_{K_A}$
- M3.** $A \rightarrow B : \{[N_B]_{K_A^{-1}}\}_{K_B}$

Tipi di Attacco

- **MitM attack:** $A \leftrightarrow Z \leftrightarrow B$, come in DH (bisogna aggiungere uno scambio di ID con messaggi firmati).
- **Replay attack:** riutilizzare parti di messaggi vecchi (o vecchie chiavi);
- **Masquerading attack:** im personificare un altro attore. Può avvenire in due modi:
 - *Z* forgia un indirizzo sorgente (nei protocolli di rete)

- Z convince altri attori che la chiave pubblica di A è K_Z (per fare *spoofing attack*);
- **Reflection attack:** reinvio di informazioni al mittente;
- **Oracle attack:** uso delle risposte previste dal protocollo come strumenti di codifica/decodifica;
- **Type flaw attack:** sostituzione di un campo con un'altra informazione che verrà interpretata dal ricevente in un modo favorevole all'attaccante.

Il Protocollo di Otway-Rees

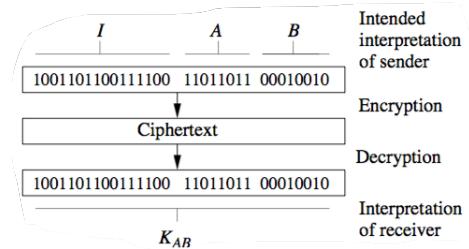
È un protocollo *server-based* per distribuzione di chiavi autenticate (con *key freshness*) ma senza autenticazione dell'entità coinvolta o *key confirmation*.

Le chiavi del server sono note e I è l'identificatore del protocollo utilizzato (un intero).

Vulnerabilità:

Assumiamo che $|\{I, A, B\}| = |\{K_{AB}\}|$, cioè se I è 32 bit, A e B sono 16 bit, allora K_{AB} è 64 bit.

- M1. $A \rightarrow B : I, A, B, \{N_A, I, A, B\}_{K_{AS}}$
M2. $B \rightarrow S : I, A, B, \{N_A, I, A, B\}_{K_{AS}}, \{N_B, I, A, B\}_{K_{BS}}$
M3. $S \rightarrow B : I, \{N_A, K_{AB}\}_{K_{AS}}, \{N_B, K_{AB}\}_{K_{BS}}$
M4. $B \rightarrow A : I, \{N_A, K_{AB}\}_{K_{AS}}$



Segretezza e autenticazione violate!

- **Attacco 1 (Reflection/type flaw):** l'attaccante Z fa un *replay* delle parti del messaggio 1 e 4 (omettendo gli step 2 e 3);

Ora A vede N_A e accetta I, A, B come *session key*!

- **Attacco 2:** Z può giocare il ruolo di S in M2 e M3 riflettendo il messaggio M2 a B .

Adesso A e B accettano la chiave sbagliata e Z può decrittare le loro comunicazioni.

Falliscono autenticazione e *secrecy*.

- M1. $A \rightarrow Z(B) : I, A, B, \{N_A, I, A, B\}_{K_{AS}}$
M4. $Z(B) \rightarrow A : I, \{N_A, I, A, B\}_{K_{AS}}$

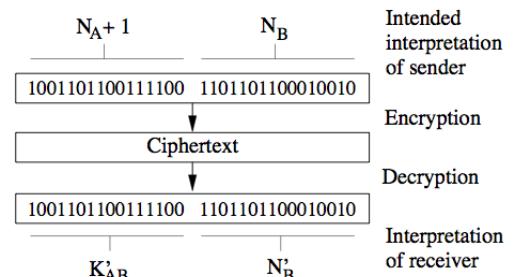
La lunghezza di I, A, B è uguale a K_{AB} ed A crede di riceverla (B in realtà non ha mai ricevuto nulla)

- M1. $A \rightarrow B : I, A, B, \{N_A, I, A, B\}_{K_{AS}}$
M2. $B \rightarrow Z(S) : I, A, B, \{N_A, I, A, B\}_{K_{AS}}, \{N_B, I, A, B\}_{K_{BS}}$
M3. $Z(S) \rightarrow B : I, \{N_A, I, A, B\}_{K_{AS}}, \{N_B, I, A, B\}_{K_{BS}}$
M4. $B \rightarrow A : I, \{N_A, I, A, B\}_{K_{AS}}$

Il Protocollo di Andrew Secure RPC

Con questo protocollo si vuole scambiare una chiave condivisa, autenticata, fresh tra due attori che condividono già una chiave simmetrica. Il protocollo stabilisce una chiave di sessione K'_{AB} e un nonce N'_B per una sessione futura.

- M1. $A \rightarrow B : A, \{N_A\}_{K_{AB}}$
M2. $B \rightarrow A : \{N_A + 1, N_B\}_{K_{AB}}$
M3. $A \rightarrow B : \{N_B + 1\}_{K_{AB}}$
M4. $B \rightarrow A : \{K'_{AB}, N'_B\}_{K_{AB}}$



Vulnerabilità: Type-flaw attack

Assumiamo che nonce e chiavi siano rappresentati da sequenze di bit della stessa lunghezza (es: 64 bit).

Z può registrare M_2 , intercettare M_3 e mandare M_2 come replay di M_4 . A a questo punto è forzato ad accettare $N_A + 1$ come chiave, la chiave **non è autenticata**. L'attacco tuttavia *non viola la segretezza*.

- M1. $A \rightarrow B : A, \{N_A\}_{K_{AB}}$
- M2. $B \rightarrow A : \{N_A + 1, N_B\}_{K_{AB}}$
- M3. $A \rightarrow Z(B) : \{N_B + 1\}_{K_{AB}}$
- M4. $Z(B) \rightarrow A : \{N_A + 1, N_B\}_{K_{AB}}$

Scambio delle chiavi con CA (Denning & Sacco)

B è certo che sta comunicando con A perché decripta con K_A che è legata al certificato C_A . Sa che il messaggio è destinato a lui perché è criptato con la sua chiave pubblica.

- K_A^{-1} chiave privata di A ;
- C_A, C_B certificati per A e B ;
- T_A timestamp generato da A , limita l'uso della session key.

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

Vulnerabilità: *MitM masquerading attack*

B crede che l'ultimo messaggio sia spedito da A , di conseguenza andrà ad utilizzare K_{AZ} , permettendo a Z di intercettare tutto.

- alice \rightarrow zoe : $C_{alice}, C_{zoe}, \{\{T_{alice}, K_{alice\ zoe}\}_{K_{alice}^{-1}}\}_{K_{zoe}}$
- zoe \rightarrow bob : $C_{alice}, C_{bob}, \{\{T_{alice}, K_{alice\ zoe}\}_{K_{alice}^{-1}}\}_{K_{bob}}$

Per difendersi da questo tipo di attacco *MitM* è necessario esplicitare i nomi degli attori nell'ultimo passo.

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

Esempi di Attacchi

• Binding attack

The protocol

- M1. $A \rightarrow S : A, B, NA$
- M2. $S \rightarrow A : S, \{S, A, NA, K_B\}_{K_S^{-1}}$

admits a binding attack:

- M1.1. $A \rightarrow Z(S) : A, B, NA$
- M2.1. $Z(A) \rightarrow S : A, Z, NA$
- M2.2. $S \rightarrow Z(A) : S, \{S, A, NA, K_Z\}_{K_S^{-1}}$
- M1.2. $Z(S) \rightarrow A : S, \{S, A, NA, K_Z\}_{K_S^{-1}}$

Fix: include the name of B in M2.

- M2. $S \rightarrow A : S, \{S, A, NA, B, K_B\}_{K_S^{-1}}$

• Parallel session attack

The one-way authentication protocol

- M1. $A \rightarrow B : \{NA\}_{K_{AB}}$
- M2. $B \rightarrow A : \{NA + 1\}_{K_{AB}}$

admits a parallel sessions attack (with 'oracle'):

- M1.1. $A \rightarrow Z(B) : \{NA\}_{K_{AB}}$
- M2.1. $Z(B) \rightarrow A : \{NA\}_{K_{AB}}$
- M2.2. $A \rightarrow Z(B) : \{NA + 1\}_{K_{AB}}$
- M1.2. $Z(B) \rightarrow A : \{NA + 1\}_{K_{AB}}$

A is forced to do work on behalf of the attacker: A acts as an 'oracle' against herself, because she provides the correct answer to her own question. At very least, A believes then that B is operational, while B may no longer exist.

Fix: add A 's name to message M1.

- M1. $A \rightarrow B : \{NA, A\}_{K_{AB}}$

• Replay attack

The Needham-Schroeder Shared Key Protocol

- M1. $A \rightarrow S : A, B, N1$
- M2. $S \rightarrow A : \{N1, B, K_{AB}, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$
- M3. $A \rightarrow B : \{K_{AB}, A\}_{K_{BS}}$
- M4. $B \rightarrow A : \{N2\}_{K_{AB}}$
- M5. $A \rightarrow B : \{N2 - 1\}_{K_{AB}}$

admits a replay attack: there is no guarantee that M3. was created by S as part of the current protocol run!

Suppose that a previously distributed key K'_{AB} has been compromised and is known to attacker Z , who monitored the network when the corresponding protocol run was executed and recorded message

- M3'. $A \rightarrow B : \{K'_{AB}, A\}_{K_{BS}}$

Z can now fool B into accepting the key K'_{AB} as new by:

- M3. $Z(A) \rightarrow B : \{K'_{AB}, A\}_{K_{BS}}$
- M4. $B \rightarrow Z(A) : \{N2\}_{K'_{AB}}$
- M5. $Z(A) \rightarrow B : \{N2 - 1\}_{K'_{AB}}$

Fix: add timestamps in relevant messages (or, as Needham-Schroeder themselves suggested, add an extra handshake at the beginning of the protocol).

• Trasformazioni crittografiche (con 2 notazioni)

- $\{M\}_K$ indica che M è criptato con chiave K , garantendo **confidentiality** poiché può essere letto solo da chi possiede la chiave K^{-1} .
- $[M]_K$ indica che M è criptato con chiave K attraverso una trasformazione *one-way*, garantendo **data integrity** con l'origine. Il messaggio può essere visto da chi possiede la chiave di decriptazione K^{-1} e la chiave K di verifica che ne controlla l'integrità (ritorna *yes* se non ci sono state violazioni).

Rappresentabile anche con $\{M\}_{K_1}^{K_2}$ denotando che M è criptato con chiave K_1 e protetto nella *data-integrity* con chiave di verifica K_2 .

Kerberos, SSL, IPSec

Kerberos

Protocollo per l'autenticazione in ambienti aperti e distribuiti. In principio doveva contenere altre due componenti: *accounting* (logging) e *audit* (registrare ciò che accade), ma non sono mai state implementate.

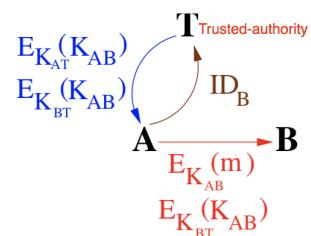
Prenderemo in esame la versione *IV* (1989), tutt'ora utilizzata da molti siti.

I **requisiti** di questo protocollo sono i seguenti:

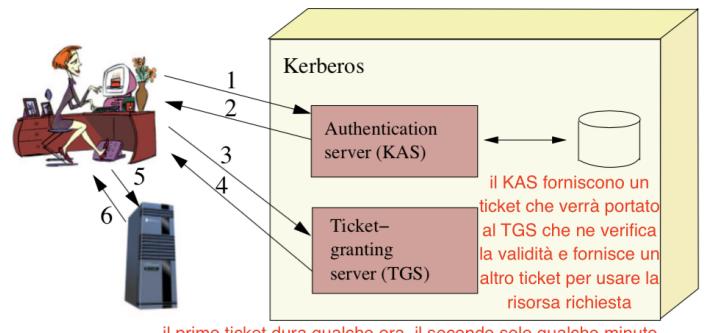
- **Sicuro**: un attaccante non deve essere in grado di impersonificare un altro utente (no *masquerading/spoofing*).
- **Affidabile**: deve supportare un architettura distribuita, dove un sistema può eseguire il backup di un altro.
- **Trasparente**: ogni utente deve usare le proprie credenziali per ottenere i servizi richiesti e non deve avere conoscenza di ciò che accade più a basso livello (*single sign-on*, cioè si registra una volta sola).
- **Scalabile**: il sistema deve supportare un gran numero di utenti e server (architettura modulare e distribuita).

Si basa approssimativamente sul protocollo *Needham-Schroeder Shared-Key*, utilizzando il **timestamp** al posto dei *nonce* per garantire la *freshness* delle chiavi.

Il *timestamp* viene creato in base all'orologio di T , dunque le altre macchine devono essere sincronizzate (un attaccante dovrebbe desincronizzarle).



- **Autenticazione** utilizza il *KAS* (*Kerberos Authentication Server*) per fornire un ticket da portare al *TGS*;
- **Autorizzazione** utilizza il *TGS* (*Ticket Granting Server*) che verifica il primo ticket e ne fornisce un altro per utilizzare la risorsa richiesta;
- **Access Control**, dove il server verifica i biglietti del *TGS* prima di fornire la risorsa richiesta.



Fase di Autenticazione

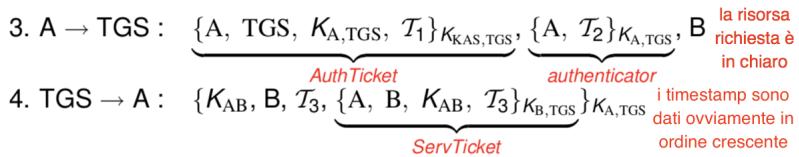
- A effettua il login e richiede una risorsa.
- KAS accede al database e manda ad A la chiave di sessione $K_{A,TGS}$ e il ticket criptato *AuthTicket*.
- A scrive la password sulla workstation per decriptare il risultato (salvato per sessione).

- Quando $K_{A,TGS}$ scade, A non è più loggato.
- $K_{A,TGS}$: chiave di sessione che dura alcune ore (dipende dall'applicazione);
- K_{AS} è derivata dalla password utente. Entrambe le chiavi devono essere registrate nel database.

1. $A \rightarrow KAS : A, TGS$
 2. $KAS \rightarrow A : \{K_{A,TGS}, TGS, \mathcal{T}_1, \underbrace{\{A, TGS, K_{A,TGS}, \mathcal{T}_1\}_{K_{KAS,TGS}}}_{AuthTicket}\}_{K_{AS}}$
- nella chiave è specificato
che riguarda l'utente A

Fase di Autorizzazione

- A presenta al *TGS* il **ticket di autenticazione** (*AuthTicket*) del passo 2. Presenta un **autenticatore** della durata di pochi secondi per evitare *replay attacks* (memorizzate anche dal server per evitare *immediate replay*);
- *TGS* risponde con una nuova chiave di sessione K_{AB} (con la durata di pochi minuti) ed un nuovo ticket **ServTicket**. $K_{B,TGS}$ è la chiave condivisa tra *TGS* e la risorsa in rete.
- il secondo ticket ha una durata più breve rispetto a quello di autenticazione perché A può interagire spesso con il *TGS* durante una giornata (mentre è loggato).



Fase di servizio

- A presenta il ticket del passo 4 con un **nuovo autenticatore** per poter accedere alla risorsa B ;
- B assicura che la richiesta è stata ricevuta (A può essere messo in attesa se la risorsa è occupata).

5. $A \rightarrow B : \underbrace{\{A, B, K_{AB}, \tau_3\}_{K_{B,TGS}}}_{ServTicket}, \underbrace{\{A, \tau_4\}_{K_{AB}}}_{authenticator}$
6. $B \rightarrow A : \{\tau_4 + 1\}_{K_{AB}}$

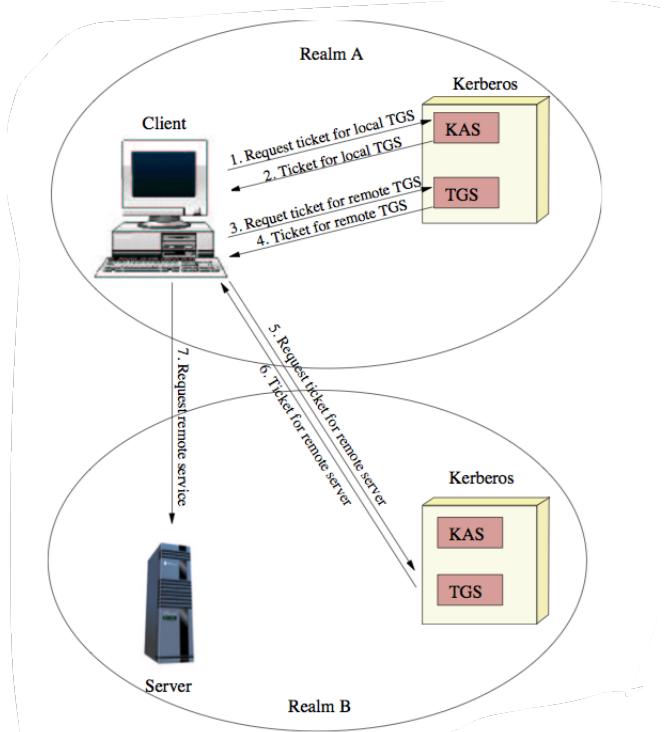
Scalabilità - Multiple Realms/Kerberi

Un **realm** è definito da un *Kerberos server*.

Le grandi reti devono necessariamente essere suddivise in *realms* amministrativi.

Kerberos supporta anche i **protocolli inter-realm**, cioè permette agli utenti di un *realm* di utilizzare risorse situate in un altro *realm*.

- i server sono registrati tra loro;
- se A vuole accedere alla risorsa B in un altro *realm*, il *TGS* del *realm* di A fornisce i ticket per accedere al *TGS* nel *realm* di B .
- l'estensione del protocollo è semplice perché vengono aggiunti solamente due nuovi passi.
- il problema riguarda la complessità: per n *realms*, nella versione *IV* il problema di distribuzione delle chiavi è $O(n^2)$.



Limitazioni di Kerberos IV

- **M1:** la crittografia non serve, però un attaccante può inondare il *KAS* (Denial of Service);
- **M2:** la doppia cifratura è ridondante (rimossa in *Kerberos V*):

$$KAS \rightarrow A : A, \{K_{A,TGS}, TGS, \tau_1\}_{K_{AS}}, \{A, TGS, K_{A,TGS}, \tau_1\}_{K_{KAS,TGS}}$$

- si basa su **clock** sincronizzati e non compromessi. Se l'host è compromesso, anche il *clock* può essere compromesso, facilitando i **replay attack**.

Alcune di queste limitazioni sono ancora presenti in *Kerberos V* che viene utilizzato come nuovo standard.

SSL (Secure Socket Layer)

Garantisce comunicazioni *end-to-end* sicure in presenza di un attaccante, usato specialmente nella navigazione web. Agisce tra client (es. *web browser*) e server (es. *website*). Usa i certificati digitali per verificare le identità degli attori e le loro chiavi pubbliche (spesso solamente il server è autenticato). Opera al di sopra del *livello di trasporto*.

Usa le chiavi pubbliche per stabilire un segreto condiviso.
SSL si basa su due protocolli:

- *Handshake*: usa la crittografia a chiave pubblica per stabilire chiavi condivise tra client e server;
- *Record*: usa la chiave segreta stabilita nella modalità *Handshake* per proteggere confidenzialità, integrità e autenticità dei dati scambiati tra client e server.



Fasi del Protocollo

• Hello

client hello $A \rightarrow B: A, Na, Sid, Pa$
server hello $B \rightarrow A: Nb, Sid, Pb$

- A identifica il client, nella pratica si utilizza l'indirizzo IP;
- Sid è l'identificatore di sessione;
- P_a è la lista delle preferenze di A per criptazione e compressione, cioè i protocolli che utilizza (es. *Diffie-Hellmann* per la signature, *RSA* per lo scambio delle chiavi);
- P_b è il protocollo con la versione maggiore, supportata da entrambi (si basa su P_a); entrambi inviati non protetti e, se la chiave privata è compromessa, l'attaccante ottiene ciò che vuole.

• Client exchange

client certificate (opt) $A \rightarrow B: \text{certificate}(A, K_A)$
client key exchange $A \rightarrow B: \{PMS\}_{K_B}$
certificate verify (opt) $A \rightarrow B: \{\text{hash}(\dots)\}_{K_A^{-1}}$

- PMS è un **pre-master secret** usato per computare un **master secret** M ; viene utilizzato per generare varie chiavi attraverso il metodo P_a e P_b concordato (viene inviato criptato con la chiave pubblica del server se utilizzano *RSA*). Es: $M = PRF(PMS, N_a, N_b)$ con PRF funzione pseudo-random (costruita dal MAC).

• Server Certificate

server certificate $B \rightarrow A: \text{certificate}(B, K_B)$

- *Certificato*: X.509v3 firmato da una terza parte affidabile; la fiducia si basa tutta sul certificato verificato dal browser.
- il server può richiedere il certificato del client, ma accade molto raramente.

• Finish

client finished $A \rightarrow B: \{\text{Finished}\}_{clientK}$
server finished $B \rightarrow A: \{\text{Finished}\}_{serverK}$

- *Finished* è un hash dei messaggi precedentemente inviati (garantendo l'integrità degli oggetti). Previene **downgrade** e **MitM attack**.
- $clientK$ e $serverK$ sono chiavi simmetriche per cifratura/decifratura di client e server; entrambe generate da N_a , N_b e M , cioè dal PMS . I messaggi seguenti saranno inviati utilizzando queste chiavi di cifratura.

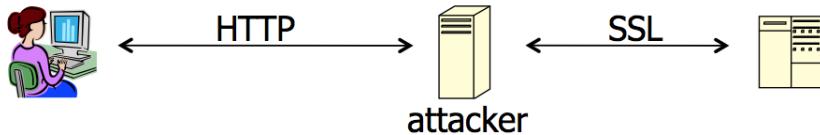
Vulnerabilità: Rollback attack

- Un attaccante potrebbe interporsi tra il client e il server, modificando la versione richiesta dal client. Questo perché il *server adatta il protocollo in base alle richieste del client*. Utilizzando la versione 2.0 di SSL, non è inclusa la fase *Finish* dei messaggi (**rollback attack** possibile perché P_a è passato in chiaro).

- I nuovi protocolli vengono introdotti perché quelli vecchi non sono più sicuri; però le nuove versioni devono essere *backward-compatibili* poiché non tutti effettuano gli upgrade.
- Una risoluzione possibile, nella fase di **client key exchange**, è inviare: $C \rightarrow S : \{version_c, secret_c\}_{PK_s}$
In questo modo, se l'**hello** viene **alterato**, il server capisce la modifica.

SSL Striping

L'**icona** del **lucchetto** nella barra di ricerca del browser viene mostrata quando la pagina è sicura da attacchi di rete.



- Nei siti in cui si passa da HTTP a HTTPS mediante javascript (*mixed content*), un attaccante può instaurare una connessione HTTPS con il server e HTTP con il client, o HTTPS con il client iniettando il proprio certificato.
N.B.: l'attacco non dovrebbe funzionare se il sito supporta HSTS (High-Strict Transport Security).
- possono essere errori di implementazione degli sviluppatori (le banche servono tutti i contenuti in *HTTPS*).
N.B.: i contenuti, nel secondo caso, sono serviti con lo stesso protocollo del resto della pagina.

<script src=http://www.site.com/script.js> </script> <script src=/www.site.com/script.js> </script>

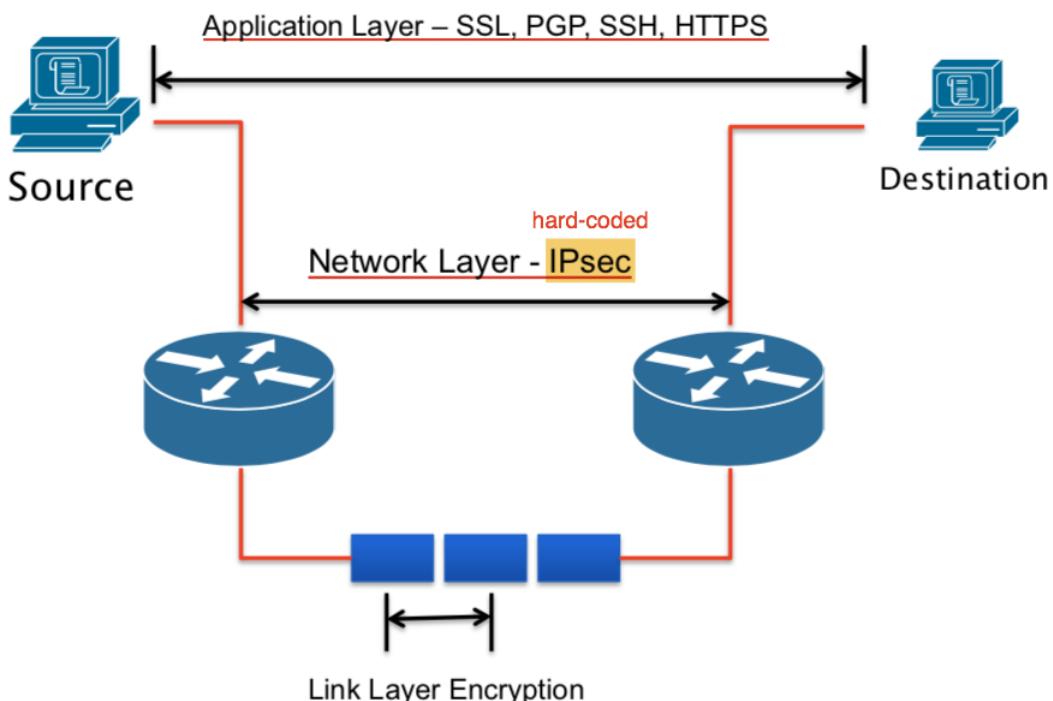
Phishing

Un altro enorme **problema** nell'ambito della sicurezza di rete è il **phishing**, nonostante esista *SSL*.

Vengono spedite moltissime mail, spacciandosi per aziende note, che richiedono di inserire dati sensibili all'interno di pagine non protette (ma l'utente inesperto non se ne accorge perché magari non distingue *HTTP* da *HTTPS*).

Una **soluzione** può essere la combinazione di *SSL* e *GSM* (telefoni cellulari), inviando un **TAN Transaction Authentication Number** via sms per l'autenticazione.

Non è troppo sicuro poiché il numero del client è salvato all'interno del server: rischio di *MitM attack*.

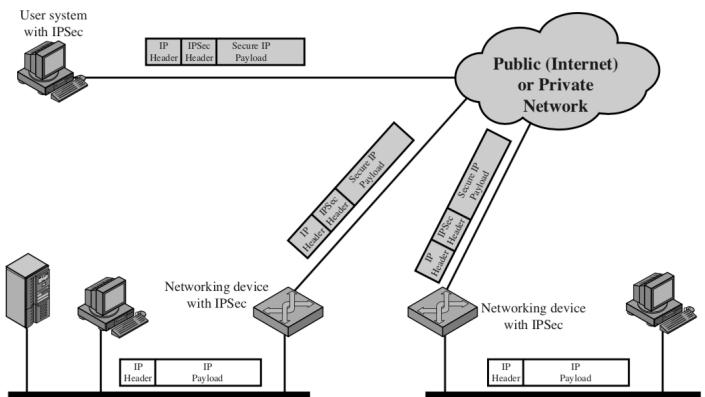


IPSec

Il protocollo internet (IP) non è sicuro: inventato agli albori di internet, quando la sicurezza non era un problema.

IP Security (IPSec) dovrebbe colmare tutte le lacune lasciate dal protocollo IP (sia IPv4 che IPv6), tra cui:

- *spoofing della sorgente*;
- *replay dei pacchetti*;
- *confidenzialità*: cifratura dei dati;
- *integrità dei dati*: tutti i router calcolano il checksum oppure l'hash dei dati;
- *autenticazione*: firma dei certificati.



Fornisce un canale sicuro a tutte le applicazioni per cifratura e autenticazione del traffico. Ha inoltre la capacità di filtrare, basandosi su una policy, come se ci fosse un *firewall* tra le due entità. È installato nei sistemi operativi (sicurezza end-to-end), nei gateway di sicurezza (router, firewall); viene utilizzato anche per implementare le *VPN* (*Virtual Private Network*, macchine remote che condividono un metodo di comunicazione come se appartenessero alla stessa rete).

IPSec è un protocollo molto complesso e, di conseguenza, molto difficile da attaccare:

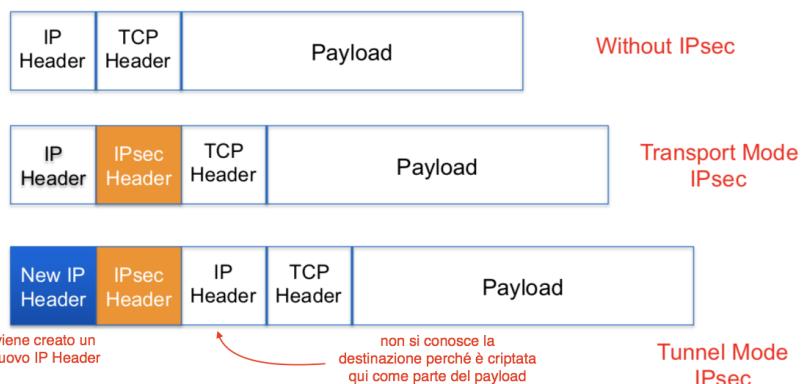
- **Authentication Header (AH)**: protegge integrità e autenticità dei datagrammi IP (non la confidenzialità);
- **Encapsulating Security Payload (ESP)**: protegge la confidenzialità e, optionalmente, l'integrità;
- **Key Management (IKE)**: *Internet Key Exchange Protocol* per la creazione del tunnel IPSec.

Security Association (SA): è una relazione unidirezionale tra mittente e destinatario. Ne servono 2 per definire i servizi di sicurezza, specificando: *authentication algorithm (AH)*, *encryption algorithm (ESP)*, chiavi, vita delle chiavi, vita della SA, modalità del protocollo (*tunnel* oppure *transport*). La SA viene stabilita utilizzando IKE (o altri protocolli).

Transport & Tunnel Mode

IPSec può lavorare in **due modalità**:

- **Transport mode**:
 - l'header IPSec viene inserito nel pacchetto IP;
 - viene criptato il payload del datagramma IP (nessun nuovo pacchetto);
 - Non si nasconde l'identità della sorgente;
 - utile per le reti in cui l'incremento della dimensione dei pacchetti può creare problemi;
 - Usato nelle VPN remote-access.
- **Tunnel mode**:
 - viene criptato l'intero datagramma IP, diventando poi parte dei dati di un nuovo pacchetto IP più grande;
 - Nasconde l'identità della sorgente;
 - Usato nelle VPN site-to-site.



Authentication Header (AH)

Viene aggiunto un nuovo header tra il livello 3 e 4 (IP e TCP), fornendo al destinatario abbastanza informazioni per identificare la SA.

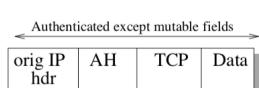
AH garantisce l'integrità e protegge parte dell'header IP. Il *sequence number* è inizializzato a zero e incrementato dal mittente per ogni pacchetto. In questo modo il destinatario può ordinarli ed eliminare i duplicati.

Il MAC del pacchetto è inserito in *Authentication Data*.

Next header	Payload length	Reserved
type of payload after AH	length of AH in 32-bit words (-'2')	for future use
Security Parameters Index (SPI)		
field identifying the SA for the datagram (value 0 indicates that no SA exists)		
Sequence number field		
counter value, used to detect replayed packets		
Authentication data		
variable number of 32-bit words containing the authentication data, e.g., a MAC (MD5 or SHA-1)		

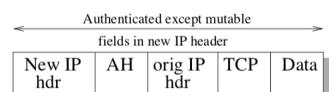
Garantisce autenticazione

- *Transport mode*:



- AH inserito dopo l'header IP e prima del payload;
- MAC dell'intero pacchetto (eccetto campi mutabili);
- garantisce una protezione end-to-end tra i sistemi abilitati IPSec.

- *Tunnel mode*:



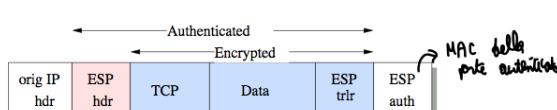
- autenticato l'intero pacchetto, con un nuovo IP header;
- l'header più interno contiene l'indirizzo dell'ultima sorgente e destinazione;
- anche il nuovo header è protetto e può contenere diversi indirizzi IP (es. firewall o gateway di sicurezza).

Encapsulating Security Payload (ESP)

32 bits	
Security Parameters Index (SPI)	
field identifying the SA for the datagram (value 0 indicates that no SA exists)	

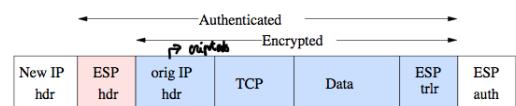
Opaque transform data	
protected field containing further parameters relevant for the processing of the cryptographic algorithm	

- *Transport mode*:

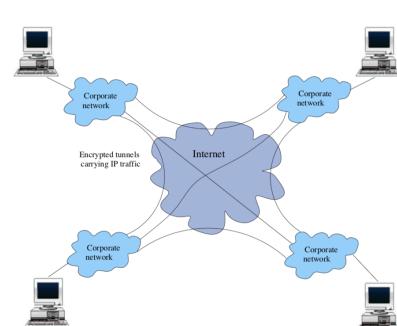
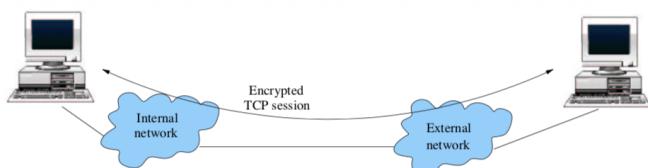


- Cripta solo il *payload* di ogni pacchetto, senza toccare l'header;
- usato per end-to-end encryption tra host che supportano IPSec.

- *Tunnel mode*:



- l'intero datagramma IP viene incapsulato nell'ESP. Il vecchio header ed il payload vengono criptati (e optionalmente autenticati);
- modalità usata per VPN



Servizi IPSec

	AH	ESP (encryption only)	ESP (encryption plus authentication)
Access control	✓	✓	✓
Connectionless integrity	✓		✓
Data origin authentication	✓		✓
Rejection of replayed packets	✓	✓	✓
Confidentiality		✓	✓
Limited traffic flow confidentiality		✓	✓

	Transport Mode SA	Tunnel Mode SA
AH	Authenticates IP payload and selected portions of IP header and IPv6 extension headers.	Authenticates entire inner IP packet (inner header plus IP payload) plus selected portions of outer IP header and outer IPv6 extension headers.
ESP	Encrypts IP payload and any IPv6 extension headers following the ESP header.	Encrypts entire inner IP packet.
ESP with Authentication	Encrypts IP payload and any IPv6 extension headers following the ESP header. Authenticates IP payload but not IP header.	Encrypts entire inner IP packet. Authenticates inner IP packet.

IKE

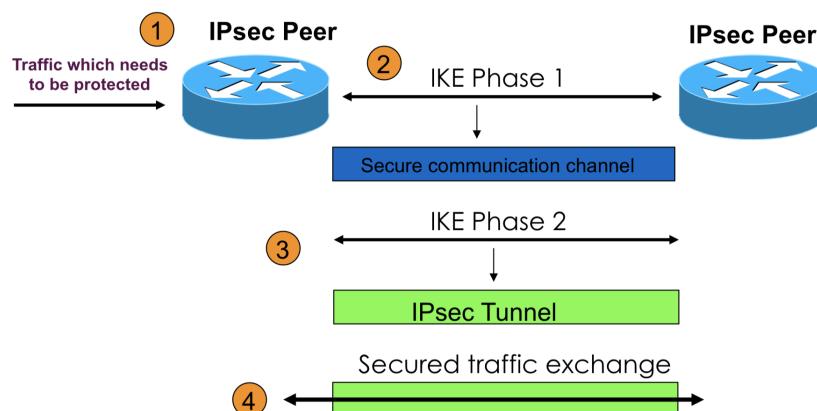
IKE è la base fondante per l'handshake in *IPSec* e si è evoluto da diversi protocolli basati su *Diffie-Hellmann*:

- *ISAKMP (Internet Security Association and Key Management)*: framework per stabilire associazioni sicure e le relative chiavi crittografiche (però era sprovvisto di meccanismi di autenticazione);
- *Oakley*: protocolli di key agreement.

IKE si basa molto su *Diffie-Hellman*, sfruttando la proprietà della **PFS** (*perfect forward secrecy*): un attaccante che intercetta la conversazione non può decriptarla neanche in seguito grazie alla *chiave di sessione temporanea*, non derivabile dalle informazioni del nodo dopo la chiusura della sessione.

Non stabilisce solo le chiavi ma anche le (SA). È un protocollo molto flessibile ma complesso; esso è composto da:

- **Fase 1**: negoziazione tra due parti di una *Security Association* (SA) attraverso le *master keys* per stabilire il materiale di codifica condiviso.
Master keys: chiavi simmetriche segrete pre-condivise, chiave pubblica di cifratura, o chiave pubblica di firma.
- **Fase 2**: la SA viene usata per creare SA figlie per autenticare e cifrare le nuove comunicazioni.

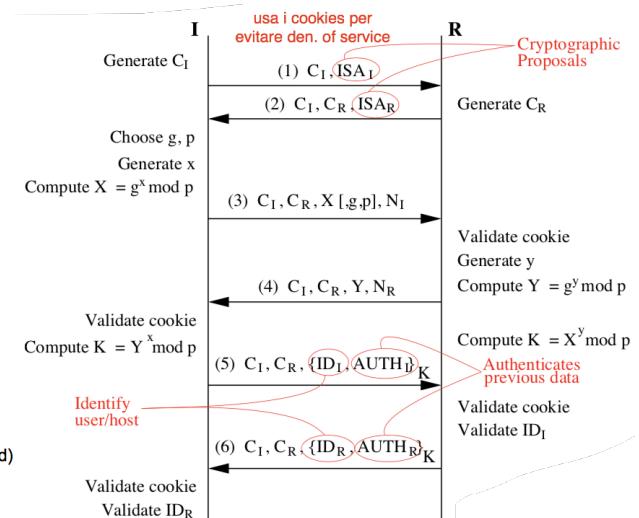
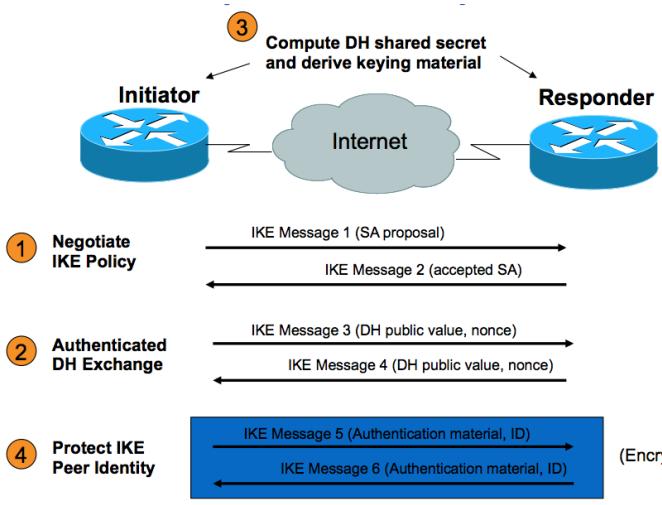


Fase 1

La fase 1 offre due modalità (entrambe risultanti nella SA desiderata):

- **Main Mode:** 6 messaggi tra I e R . Offre protezione dell'identità e flessibilità nella negoziazione dei parametri.
- **Aggressive Mode:** più veloce ma meno protezione. Si basa su 3 messaggi scambiati. Non ha protezione dell'identità tranne quando viene usata la crittografia a chiave pubblica per l'autenticazione.

Ogni modalità ha 4 varianti, basati sul metodo di autenticazione (chiave condivisa, firme digitali e due varianti a chiave pubblica). Prendiamo in esame la **Main Mode**:



A chiave condivisa

- (1) $I \rightarrow R : C_I, ISA_I$
- (2) $R \rightarrow I : C_I, C_R, ISA_R$
- (3) $I \rightarrow R : C_I, C_R, X [g, p], N_I$
- (4) $R \rightarrow I : C_I, C_R, Y, N_R$
- (5) $I \rightarrow R : C_I, C_R, \{ID_I, AUTH_I\}_K$
- (6) $R \rightarrow I : C_I, C_R, \{ID_R, AUTH_R\}_K$

- 1: ISA_I (ISAKMP SA per I) è il numero di proposte crittografiche con una lista di algoritmi; Viene inviato un *cookie* per evitare *DoS/Clogging* in caso di elevato numero di richieste dallo stesso IP.

- 2: ISA_R (ISAKMP SA per R) è il numero di proposte crittografiche accettate e gli algoritmi scelti; ritorna anche un altro *cookie*, oltre a quello del mittente.

- 3 e 4: contengono le half-key per DH, i nonce di I ed R ed i rispettivi *cookie*.

- 5 e 6: sono crittografati con la chiave generata da Diffie-Hellman. $AUTH_I$ e $AUTH_R$ sono l'autenticazione dei messaggi precedenti attraverso un MAC o una firma.

A chiave pubblica

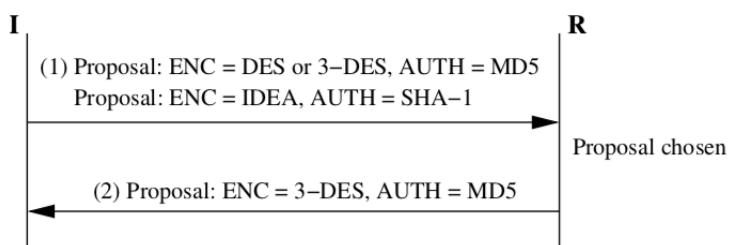
Concrete example of authentication variant using digital signatures:

(1) $I \rightarrow R : C_I, ISA_I$	
(2) $R \rightarrow I : C_I, C_R, ISA_R$	
(3) $I \rightarrow R : C_I, C_R, g^x, N_I$	
(4) $R \rightarrow I : C_I, C_R, g^y, N_R$	
(5) $I \rightarrow R : C_I, C_R, \{ID_I, SIG_I\}_{SKYID_e}$	
(6) $R \rightarrow I : C_I, C_R, \{ID_R, SIG_R\}_{SKYID_e}$	

combinazione di 2 chiavi (K nelle slide precedenti)

$SKEYID = h(\{N_I, N_R\}, g^{xy})$ (h is keyed hash)	<i>deriving key</i>
$SKEYID_d = h(SKEYID, \{g^{xy}, C_I, C_R, 0\})$	<i>authentication key</i>
$SKEYID_a = h(SKEYID, \{SKEYID_d, g^{xy}, C_I, C_R, 1\})$	<i>encryption key</i>
$SKEYID_e = h(SKEYID, \{SKEYID_a, g^{xy}, C_I, C_R, 2\})$	
$HASH_I = h(SKEYID_a, \{g^x, g^y, C_I, C_R, ISA_I, ID_I\})$	
$HASH_R = h(SKEYID_a, \{g^y, g^x, C_R, C_I, ISA_R, ID_R\})$	
$SIG_I = \{HASH_I\}_{K_I^{-1}}$	
$SIG_R = \{HASH_R\}_{K_R^{-1}}$	

Negoziazione fallita



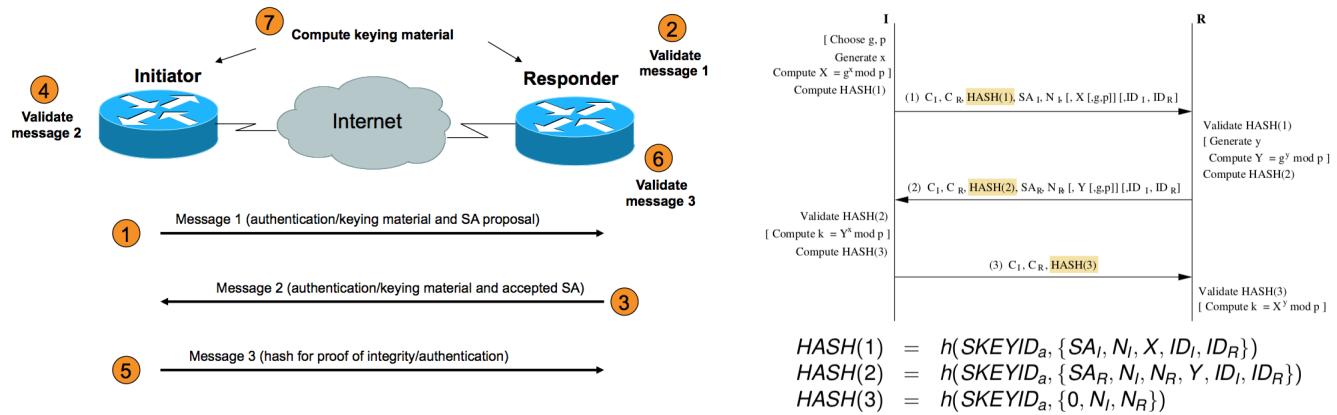
L'Aggressive Mode si basa sui seguenti 3 messaggi:

- il mittente propone la SA, inizia lo scambio DH, manda un numero pseudorandomico e la propria identità IKE;
- il destinatario accetta la SA, autentica il mittente, manda un numero pseudorandomico, la sua identità IKE e, se possiede un certificato, lo manda;
- il mittente autentica il destinatario, conferma lo scambio e, se possiede un certificato, lo manda.

Fase 2 (Quick mode)

Tutto il traffico viene criptato utilizzando la *ISAKMP Security Association* poiché è stato creato un canale sicuro. Ogni negoziazione *quick mode* risulta in due *IPSec Security Association* per mettere in sicurezza i dati nel tunnel IPSec.

- Vengono scelti algoritmi di cifratura e autenticazione per i dati trasmessi.
- Si scegli il protocollo di sicurezza (AH oppure ESP).
- Crea e aggiorna le chiavi sfruttando la PFS di Diffie-Hellman.



$$\begin{aligned}
 \text{HASH}(1) &= h(SKEYID_a, \{SA_I, N_I, X, ID_I, ID_R\}) \\
 \text{HASH}(2) &= h(SKEYID_a, \{SA_R, N_I, N_R, Y, ID_I, ID_R\}) \\
 \text{HASH}(3) &= h(SKEYID_a, \{0, N_I, N_R\})
 \end{aligned}$$

Intrusion Detection System

Un problema significativo per sistemi di rete sono gli **accessi indesiderati** o **ostili**.

Ci sono diverse classi di **intruders** (oppure *hackers/crackers*):

- **Masquerader**: un individuo non autorizzato a utilizzare il computer (*outsider*);
- **Misfeasor**: utente legittimo che accede in maniera non autorizzata a dati, risorse o abusa dei privilegi (*insider*);
- **Clandestine User**: individuo che prende il controllo del supervisore del sistema per eludere *access control* e *audit*.

Gli *intruders* possono essere **benigni** (esplorano la rete e consuma risorse rallentando le performance) oppure **maligni** (accedono ai dati modificandoli e corrompendo il sistema). *IDS/IPS/VPN* possono essere d'aiuto.

Il **comportamento di un hacker** segue dei passi ben precisi:

- sceglie un target attraverso dei tools di *IP lookup*;
- mappa la rete per trovare servizi accessibili;
- identifica i servizi potenzialmente vulnerabili;
- può eseguire un *brute force attack* per ottenere delle password;
- può installare nel target dei tool per l'amministrazione remota;
- può attendere che l'admin effettui il login per catturarne la password;
- utilizza la password per accedere al resto della rete.

Criminal Enterprise: gruppi di hacker organizzati con un obiettivo preciso. Quando penetrano agiscono in velocità ed escono. Per questo motivo i dati sensibili hanno bisogno di una maggiore sicurezza. Agiscono seguendo dei passi:

- commettono pochissimi errori.
- sfruttano le porte vulnerabili;
- usano degli *sniffers* per catturare password;
- usano *trojan horse* nascosti per lasciare un'entrata nascosta da cui rientrare;
- agiscono velocemente ed in maniera precisa per rendere le loro attività difficili da individuare;

Insider Attacks: tra i più difficili da rilevare e prevenire. Gli impiegati hanno accesso e conoscono il sistema. Possono avere diversi motivi per fare danni (es. licenziamenti). *IDS/IPS* possono aiutare ma hanno bisogno di: meno privilegi, monitoraggio dei logs, strong authentication e processi di terminazione per bloccare gli accessi. Agisce come segue:

- crea account di rete per sé e per i loro amici;
- accede ad account e applicazioni che normalmente non utilizza per lavori quotidiani;
- conduce conversazioni furtive;
- visita siti web che si rivolgono a impiegati scontenti;
- effettua molti download e copie di file;
- accede alla rete durante le ore di riposo.

Intrusion Techniques: ottenere l'accesso e/o incrementare i privilegi nel sistema. Sfrutta le vulnerabilità del sistema o di alcuni software. L'obiettivo principale è quasi sempre ottenere delle password per impersonificare un altro utente.

- **Password Guessing**: l'attaccante conosce un login (da email, pagina web ...) e prova a indovinare la password da informazioni sull'utente. Il successo dipende dalla password scelta dall'utente (spesso poco sicura).
- **Password Capture**: spiare l'immissione da parte dell'utente, utilizzare un *trojan horse*, monitoraggio di network login non sicuro, estrarre informazioni dalla cronologia web e dalla cache.

Intrusion Detection: è inevitabile avere delle fallo nel sistema di sicurezza, però è necessario rilevare le intrusioni nel sistema, bloccarle e raccogliere informazioni per migliorare la sicurezza. Un intruso agisce in modo diverso dagli altri utenti legittimi all'interno del sistema; è comunque molto difficile da identificare.

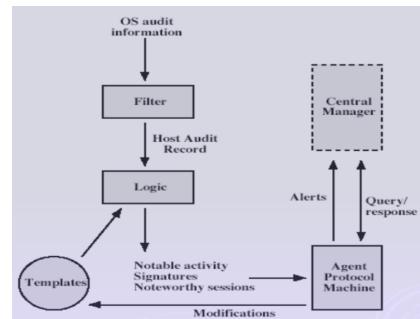
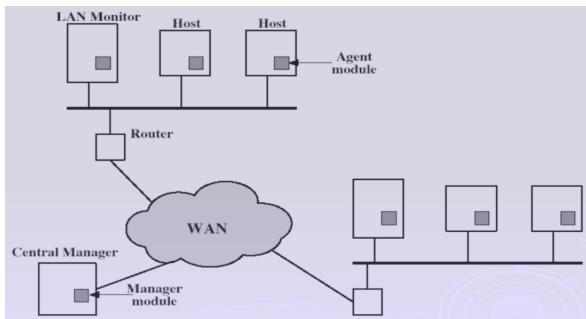
Esistono principalmente due approcci per la *Intrusion Detection*:

- **Statistical Anomaly Detection** (*profile of the user*): definisce i comportamenti aspettati da un utente e verifica se ci sono devianze da esso. Definisce anche una soglia di rilevamento (*threshold detection*) per contare le occorrenze di un evento nel tempo e valutare se è un valore ragionevole oppure c'è un'anomalia.
- **Rule-based Detection** (*profile of the attack*): definisce i modelli dell'attacco per il rilevamento di anomalie; si ricercano inoltre dei comportamenti sospetti (*penetration identification*). Osserva gli eventi nel sistema e applica i modelli (di attacchi noti) per decidere se un'attività è sospetta o meno.
- **Audit Records**: sono un tool fondamentale per l'*intrusion detection* e possono essere nativi oppure specifici (costo maggiore ma sono creati specificatamente per collezionare le informazioni desiderate).

Un *Intrusion Detection System* deve rilevare una elevata percentuale di intrusioni con il minor numero di falsi allarmi; se rileva poche intrusioni non è sicuro, se ci sono troppi falsi allarmi viene ignorato (perdita di tempo).

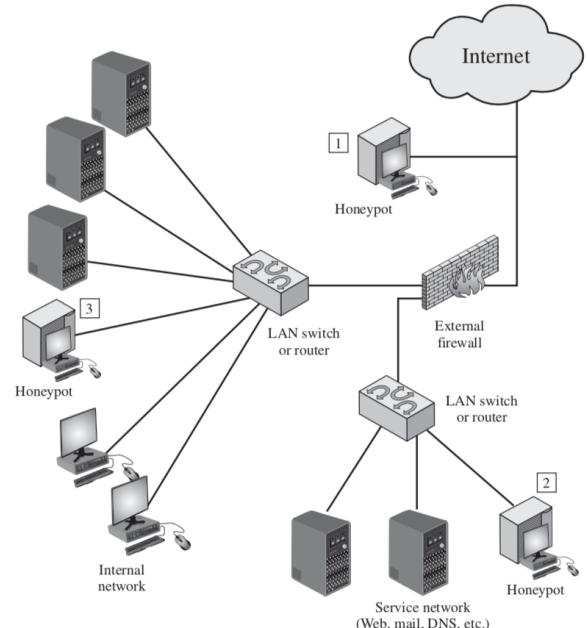
Distributed Intrusion Detection: è una difesa più effettiva che si occupa di più sistemi connessi ad una rete.

Problemi: diversi formati degli *audit record*, integrità e confidenzialità dei dati in rete, architettura centralizzata o meno.



Honeypots: sistemi serviti come esche per adescare l'attaccante; sembrano interessanti per le informazioni che cerca un attaccante. Lo incoraggiano a restare nel sistema per catturare informazioni sulle sue attività.

- *Location 1*: situato all'esterno, non può attrarre attaccanti interni, soprattutto se il firewall filtra il traffico in entrambe le direzioni.
- *Location 2*: situato nella rete di servizi esterni (*DMZ, Demilitarized Zone*); i sistemi in quella rete sono sicuri però il firewall blocca alcuni accessi a quella *network area*. Perciò spesso il firewall deve aprire maggiormente al traffico, aumentando i rischi.
- *Location 3*: situato all'interno, può rilevare attacchi interni e anche firewall configurati male che collegano internet alla rete interna; se l'*honeypot* è compromesso, ci possono essere attacchi interni.

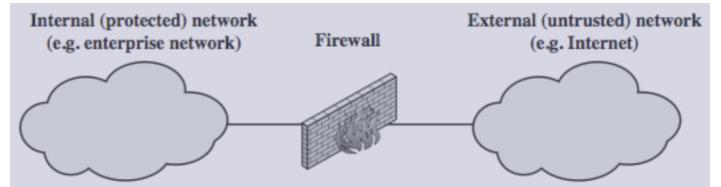


Password Management: è la prima linea di difesa contro gli intrusi. Utilizzano il *login* (per determinare i privilegi dell'utente) e la *password* per identificarsi. Quest'ultima viene salvata criptata (con DES o funzioni di hash crittografico). Gli utenti solitamente generano password deboli; è necessario educarli riguardo l'importanza di una buona password. Le password più sicure sono quelle generate dal computer: difficili, mai utilizzate da altri, complesse da memorizzare.

Proactive Checking: permette all'utente di scegliere la password ma il sistema verifica se è accettabile (compara con un dizionario di password cattive; usa algoritmi per rilevare scelte deboli).

Firewall

Tutte le organizzazioni, e agenzie governative vogliono essere connesse alla rete e interconnettere tutte le reti. Non è semplice mettere in sicurezza tutti i sistemi dell'organizzazione: di solito si utilizzano i **Firewall** per avere una difesa perimetrale con un singolo *choke point*.



Il *Firewall* è un *choke point* per il controllo e monitoraggio che interconnette reti con diverso livello di *trust*. Impone restrizioni ai servizi di rete, consentendo solo il traffico autorizzato: effettua un controllo sugli accessi e può fornire allarmi riguardo comportamenti anormali (fornisce una *NAT*). Deve essere impenetrabile.

Limitazioni: non può proteggere da attacchi che lo bypassano (*trusted organization and services*), attacchi interni, accesso via *WLAN* e malware importati da altri dispositivi.

Tipi di Firewall:

- **Packet Filters:** esamina ogni pacchetto IP e ne consente o meno l'accesso alla rete seguendo le regole (controlla *source/dest IP*, protocollo IP ...). Limita l'accesso ai servizi (porte) e segue una *policy* di default (ciò che non è esplicitamente permesso è proibito, ciò che non è esplicitamente proibito è permesso).

Attacchi ai Packet Filters:

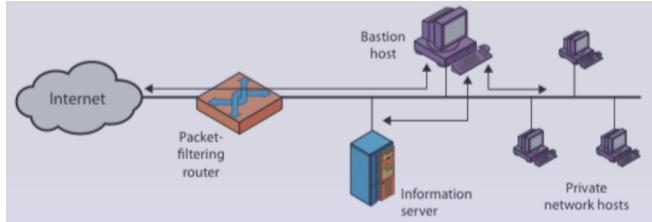
- *Spoofing degli indirizzi IP*: servono filtri per pacchetti con indirizzo sorgente falso (l'intruso trasmette un pacchetto dall'esterno con IP di origine di un host interno ed arriva sull'interfaccia esterna del Firewall).
- *Source routing attack*: l'attaccante imposta una *route* diversa da quella di default per evitare controlli di sicurezza; è necessario bloccare i pacchetti con *source routed*.
- *Tiny fragment attack*: le informazioni vengono suddivise in tanti piccoli pacchetti poiché, solitamente, il firewall controlla solo il primo; è necessario accettare pacchetti con una dimensione minima.
- **Stateful Packet Filters**: i *packet filters* tradizionali non esaminano i pacchetti fino agli strati più alti. Gli *stateful packet filters* esaminano ogni pacchetto IP tenendo traccia della sessione client-server e controllano la validità di ciascun pacchetto. Mantiene una directory con ogni connessione TCP aperta e permette solo il traffico di queste.
- **Application Level Gateway (or Proxy)**: ha piena conoscenza del protocollo. L'utente avanza la richiesta al proxy che la valida, la esegue e ritorna all'utente il risultato della richiesta. Controlla il traffico a livello applicativo. È necessario separare i proxy per ogni servizio: alcuni lo supportano naturalmente, altri sono più problematici. Lo svantaggio è l'*overhead* poiché deve monitorare il traffico in entrambe le direzioni.
- **Circuit Level Gateway**: trasmette due connessioni TCP. Impone la sicurezza limitando quali connessioni sono ammesse. Una volta create, trasmette il traffico senza analizzare il contenuto. Viene usato quando gli utenti interni fidati vogliono effettuare una connessione esterna.
L'utente esterno deve collegarsi al *SOCKS (SOCKET Secure)* nella porta 1080 (solitamente) e inviare la richiesta. I pacchetti vengono instradati tra client e server, passando dal proxy.

Bastion Host: si pone tra la connessione internet pubblica e la rete privata, filtrando tutti i contenuti scambiati: in caso di attacco il sistema lo blocca impedendo l'accesso alla rete locale. Questo computer ospita generalmente una singola applicazione (es. *circuit/application level gateway*), mentre tutti gli altri servizi non essenziali vengono rimossi o limitati per ridurre al minimo la minaccia di infezione del sistema stesso.

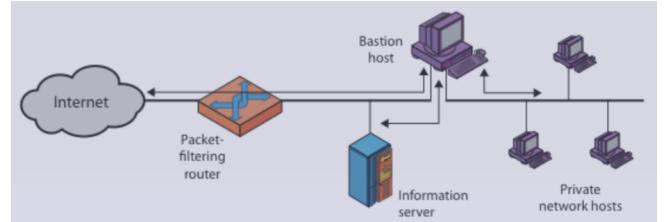
Host-Based Firewall: modulo software per rendere sicuro un host individuale (disponibile in molti OS oppure aggiungibile con package); viene spesso usato dai server. Aggiunge un ulteriore livello di protezione personalizzando le regole di filtraggio per l'ambiente dell'host.

Personal Firewall: controlla il traffico tra il PC/workstation e Internet/rete dell'impresa. È un modulo software nel computer/ISP router, DSL dell'ufficio. Solitamente è molto meno complesso degli altri firewall. Il suo ruolo principale è quello di evitare accessi remoti non autorizzati al computer e controlla le attività in uscita dovute a malware.

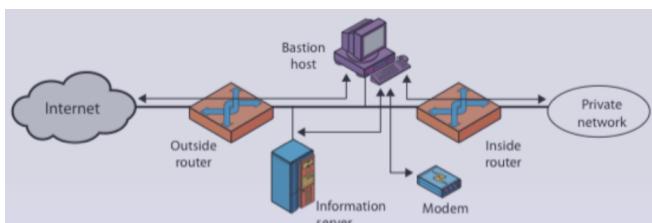
Firewall Configurations



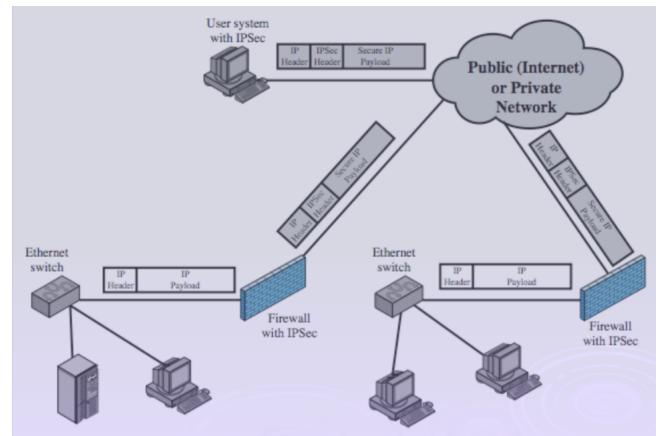
Host firewall system: single-homed bastion host.



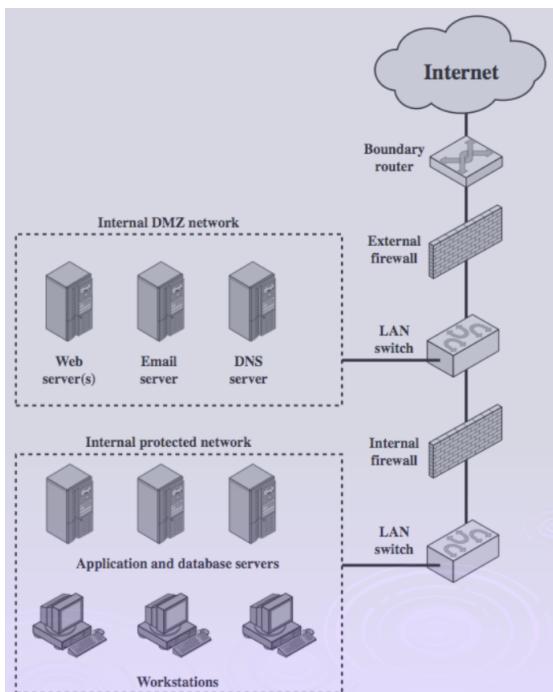
Host firewall system: dual-homed bastion host.



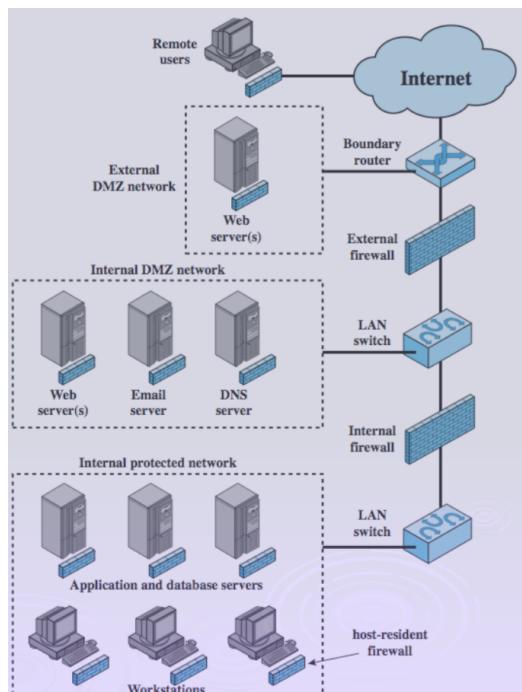
Subnet firewall system.



Virtual Private Networks (VPNs).



Demilitarized Zone (DMZ) Networks.



Distributed Firewalls.