Università degli Studi di Verona

Sicurezza delle reti

Riassunto dei principali argomenti

Davide Bianchi

April 1, 2020

Contents

1 Introduzione

Definizione 1.0.1 (Information Security) *Protezione delle informazioni e dei sistemi per impedirne l'accesso non autorizzato, uso, divulgazione, modifica o distruzione.*

Definizione 1.0.2 (Network Security) *Protezione dell'accesso a risorse situate all'interno di una rete.*

Nella sicurezza si distinguono una **policy**, un **meccanismo** e una **compliance**. Una security policy specifica il comportamento che il sistema può o non può assumere. I meccanismi di sicurezza sono l'implementazione di una data policy. Diciamo quindi che una security policy ϕ deve rimanere valida per un sistema P in ogni ambiente malevolo E, ovvero $P \parallel E \models \phi$.

Le politiche di sicurezza sono spesso formulate per arrivare ad alcune proprietà standard, le più comuni sono:

- Confidenzialità: non ci sono fughe di informazioni;
- Integrità: non ci sono modifiche alle informazioni;
- Disponibilità: non ci sono "danneggiamenti" ai servizi;
- Accountability[1]: le azioni sono sempre riconducibili ai diretti responsabili;
- Autenticazione: l'origine dei dati può essere identificata con sicurezza.

Contromisure per la protezione. Le principali tecniche di contromisura consistono in:

- Prevenzione di breach;
- Rilevamento di attacchi in corso;
- Reazione ad un possibile attacco.

[1] La traduzione più vicina è *responsabilità*.

2 Cenni di crittografia

2.1 Introduzione

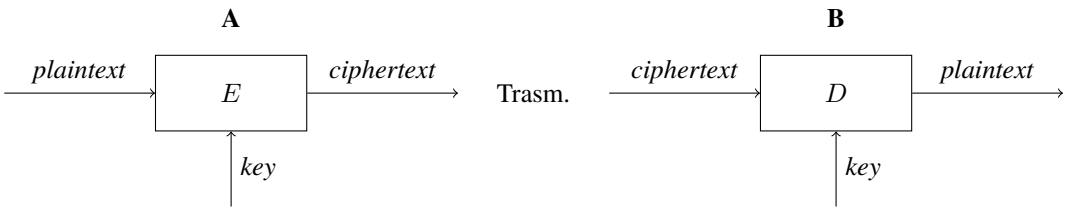
Iniziamo dando alcune definizioni fondamentali. Si useranno i termini *ciphertext* e *plaintext* per indicare rispettivamente il testo cifrato e quello in chiaro.

Definizione 2.1.1 (Crittografia) *Insieme dei metodi per rendere un messaggio non leggibile ad altri.*

Definizione 2.1.2 (Steganografia) *Insieme dei metodi per nascondere l'esistenza di un messaggio in un altro contenuto.*

Definizione 2.1.3 (Crittoanalisi) *Analisi del ciphertext per ottenere il plaintext corrispondente.*

Un generico sistema crittografico è strutturato come:



In crittografia si distinguono le due categorie *a chiave simmetrica* e *a chiave asimmetrica*. La differenza sta nel fatto che nella crittografia a chiave simmetrica le due entità che si scambiano il messaggio devono condividere una stessa chiave (che deve essere trasmessa su un canale sicuro), mentre nella crittografia a chiave asimmetrica le chiavi sono differenti e sono 2 per ogni entità, una pubblica e una privata. Nella crittografia a chiave asimmetrica si elimina il problema della condivisione della chiave; inoltre la chiave pubblica può essere compromessa da attaccanti senza che la chiave privata venga compromessa, e senza che venga compromessa la segretezza del messaggio.

Un altro aspetto fondamentale della crittografia è che la cifratura e la decodifica sono facili, *se le chiavi sono note*. Da ciò consegue che la sicurezza debba risiedere nella chiave, non nell'algoritmo in se.

2.2 Crittoanalisi

La scienza di recuperare il messaggio in chiaro senza conoscere il ciphertext si basa sostanzialmente su due differenti approcci:

- attacco brute-force;
- attacco crittoanalitico.

Attacco brute-force. Un attacco bruteforce è semplice: consiste nel provare tutte le chavi possibili fino ad indovinare quella corretta. Questa tipologia di attacco in generale è sempre possibile nella sua semplicità, tuttavia, se la dimensione dello spazio delle chiavi inizia ad essere elevata, il tempo che si deve impiegare diventa insostenibile, per cui in questi casi è necessario ricorrere ad altri stratagemmi.

Attacco crittoanalitico. In questo caso si assume che l'attaccante conosca l'algoritmo utilizzato nella cifratura dei messaggi; si trova quindi una qualche debolezza nell'algoritmo che permetta di farlo fallire.

In tal senso, si tende a rendere noto un algoritmo affinchè il maggior numero di persone tenti di attaccarlo, per aumentare al massimo le possibilità che venga trovata una falla. (In contrasto con la cosiddetta **security by obscurity**).

Tipologie di attacco. Consideriamo ora i possibili attacchi che un sistema crittografico deve affrontare per essere affidabile:

- *known cypertext attack:* questo attaccante è il meno aggressivo e conosce solamente il testo cifrato;
- *known plaintext attack:* conosce entrambi i tipi di testo;

- *choosen plaintext:* può scegliere il plaintext da codificare e analizzare il ciphertext ottenuto;
- *adaptive choosen plaintext:* può liberamente scegliere il plaintext da far codificare e comportarsi di conseguenza, sulla base del risultato appena ottenuto.
- *chosen ciphertext:* l'attaccante può scegliere differenti ciphertext e avere accesso al plaintext decriptato, per infine ricavare la chiave.

2.3 Notazione

La notazione usata è la seguente:

- $\mathcal{A}$ è l'alfabeto, ovvero un insieme finito di simboli;
- $\mathcal{M} \subseteq \mathcal{A}^{*}\star$ è il messaggio, dove $M \in \mathcal{M}$ è il *plaintext*;
- $\mathcal{C}$ è il messaggio cifrato, il cui alfabeto può anche differire da quello usato per M;
- $\mathcal{K}$ indica lo spazio delle chiavi;
- ogni $e \in \mathcal{K}$ denota una funzione biiettiva da $\mathcal{M}$ a $\mathcal{C}$, viene indicata con E_e ed è la funzione di cifratura;
- ogni $d \in \mathcal{K}$ è una funzione biiettiva da $\mathcal{C}$ a $\mathcal{M}$, indicata con D_d ed è la funzione di decodifica.

Data la notazione soprastante, indichiamo con cifrario un insieme $\{E_e \mid e \in \mathcal{K}\}$ e il suo corrispondente $\{D_d \mid d \in \mathcal{K}\}$ tale che per ogni $e \in \mathcal{K}$ esiste un solo $d \in \mathcal{K}$, in modo tale che $D_d = E_e^{-1}$. La coppia $\langle e, d \rangle$ forma una coppia di chiavi, dove e e d possono anche essere identiche (come nel caso della crittografia simmetrica).

3 Crittografia a chiave simmetrica

Nella crittografia a chiave simmetrica le chiavi sono le stesse ($e = d$), e i due interlocutori condividono una chiave. I cifrari possibili nel caso della crittografia simmetrica sono di 3 categorie:

- *cifrari a blocchi:* dividono il testo in blocchi di lunghezza fissa e cifrano un blocco alla volta;
- *cifrari a flusso:* cifrari a blocchi in cui la dimensione di ogni blocco è fissata a 1;
- *codes:* cifrari che lavorano su parole a lunghezza variabile.

3.1 Tecniche di sostituzione

Sono tutti quei cifrari che sostituiscono una lettera con un'altra lettere, basandosi su una qualche regola di sostituzione, come il cifrario di Cesare e la permutazione casuale.

Cifrario di Cesare. Il messaggio viene cifrato sostituendo ogni lettera l del messaggio con la $l + k$ esima lettera dell'alfabeto; la chiave quindi è data dalla coppia $(l, l + k)$.

Il cifrario di Cesare è facile da attaccare in quanto basta un attacco *bruteforce*, quindi è sufficiente provare tutte le combinazioni (che sono in totale 26).

Permutazione casuale. Supponiamo di usare come cifrario una permutazione casuale dell'alfabeto, ovvero sostituendo ad ogni lettera dell'alfabeto un'altra lettera, in modo totalmente casuale. In tal caso l'attacco bruteforce richiederebbe tempo eccessivo (ci sono 26! pssibili combinazioni da provare, che sono decisamente troppe).

La tecnica usata per attaccare questo tipo di crittografia è l'*analisi delle frequenze*, ovvero l'analisi delle lettere che capitano di più in una data lingua, e associare la lettera del messaggio cifrato con una data frequenza con quella nella lingua del messaggio con una frequenza simile.

Cifrario di Vigenère. Il cifrario di Vigenère riprende l'idea del cifrario di Cesare. L'idea è la seguente: presa una chiave (es. *key*), si ripete la chiave tante volte quanto è lungo il testo (eventualmente troncando l'ultima ripetizione), e si codifica la lettera con il corrispondete cifrario di Cesare.

K E Y K E Y K E Y K E Y
P R O V A D I T E S T O

La prima lettera del cipher text sarà la lettera ottenuta dal cifrario di Cesare di chiave (K, P), la seconda con la chiave (E, R) e così via.

Anche questo cifrario è semplice da attaccare, si parte dalla divisione del ciphertext in gruppi di lunghezza pari a quella della chiave, e si esegue l'analisi delle frequenze su ogni gruppo.

3.2 Cifrari a trasposizione

Funzionano in maniera leggermente diversa. Dato un blocco di lunghezza l e $\mathcal{K}$ un insieme di permutazioni su $\{1 \dots t\}$, si ha che

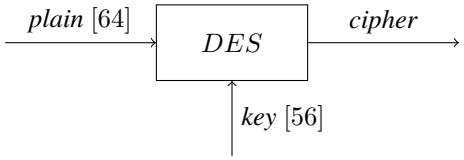
$$E_e(m) = m_{e(1)}, \dots, m_{e(2)}$$

Per decodificare si applica la permutazione inversa ad ogni carattere del ciphertext.

3.3 Cifrario di Feistel

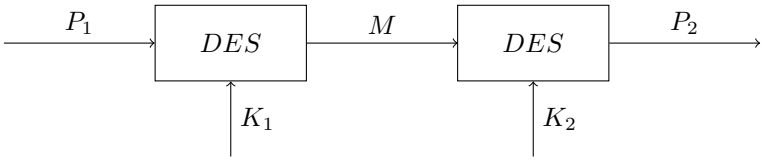
3.4 Data Encryption Standard (DES)

È un cifrario a blocchi che lavora su blocchi di 64 bit. Fu in effetti il primo standard di crittografia, e ne furono rilasciate versioni aggiornate che lavorassero su chiavi di lunghezza maggiore (3-DES). Non è ancora stato violato, ma è possibile ridurre in tempo lineare lo spazio delle chiavi da 2^{56} a 2^{43}.



3.4.1 Double DES e 3-DES

La variante DD, che usa DES due volte consecutive, è soggetta ad un attacco del tipo *Meet-in-the-Middle*.



L'attacco funziona nel seguente modo:

1. Dato un $C = E_{K_2}(E_{K_1}(P))$, sia $X = E_{K_1}(P) = D_{K_2}(C)$;
2. Dati P e C, cifrare P per ogni possibile chiave (sono 2^{56});
3. Generare una tabella con tutti i risultati, ordinati secondo X;
4. Decifrare C con tutte le possibili K_2, cercando un matching con quelle i risultati ottenuti prima. Ogni coppia è una possibile coppia valida, basta confrontare i risultati con P e C iniziali.

In effetti, guardando lo schema sopra, si nota che al massimo occorrono 2^{56} operazioni per violare questo protocollo.

Con 3-DES invece, un tentativo con la soluzione bruteforce necessita di almeno 2^{112} operazioni, un numero notevolmente più alto. Al momento infatti non esistono soluzioni per violare 3-DES.

3.5 Advanced Encryption Standard (AES)

Proposto come rimpiazzo di DES nel 1991, fu selezionato nel 2001. Infatti il DES iniziava a non andare più bene, in larga parte perchè era disegnato per i software degli anni 70 ed era abbastanza lento. AES funziona in maniera più snella e lavora su chiavi molto più lunghe (128, 192 e 256 bit).

3.6 Cipher block chaining

Come ci si comporta quando la lunghezza del messaggio eccede la dimensione del blocco? In tal caso, ci sono molteplici possibilità:

1. Splittare il messaggio in m blocchi, e cifrarli individualmente. Questa opzione è soggetta a pesanti limtazioni, la prima data dal fatto che identici plaintext vengono mappati su identici ciphertext (*information leak*); la seconda invece limita di parecchio le possibilità di individuare eventuali manomissioni del messaggio da parte di terzi (*integrity*);
2. Si può pensare in alternativa di far dipendere un carattere del ciphertext da quello precedente: dato un valore iniziale, il successivo carattere sarà cifrato con uno XOR tra il carattere precedentemente cifrato e il carattere da cifrare. In sostanza, dato un certo C_0,

$$C_i = E_K(P_i \oplus C_{i-1})$$
$$P_i = C_{i-1} \oplus D_K(C_i) \quad \text{(per la decodifica)}$$

Con la seconda soluzione, i caratteri cifrati dipendono strettamente da quelli precedenti, quindi è impossibile che due plaintext uguali vengano mappati su ciphertext uguali.

3.7 Posizionamento dei sistemi crittografici

Distinguiamo i casi di *link encryption* e *end-to-end encryption*. Nella link encryption ci sono sistemi di cifratura ad ogni collegamento, quindi i dati vengono decifrati e cifrati ad ognuno dei singoli collegamenti. Nella crittografia end-to-end invece, ciò che accade è che i sistemi di cifratura sono posizionati all'origine e alla destinazione dei dati, però in tal caso è necessario l'utilizzo di chiavi condivise tra i due interlocutori.

Guardando la questione da un punto di vista alternativo, ossia quello dello stack OSI, possiamo osservare come la link encryption sia applicata ai livelli più bassi dello stack, mentre man mano si sale viene applicata una crittografia di tipo end-to-end. Idealmente, serve che la crittografia end-to-end protegga i dati contenuti nei pacchetti, ma che lasci inalterati gli header, per permettere l'inoltro dei pacchetti. La link encryption protegge invece i dati di inoltro da monitoraggio e analisi da parte di terzi.

3.8 Distribuzione delle chiavi

Come già detto in precedenza, la crittografia a chiave simmetrica richiede che le parti condividano una chiave. Ciò può costituire un problema, dal momento che terzi malintenzionati possono sempre tentare di rubare la chiave sfruttando qualche falla nel sistema di condivisione della stessa.

Tipicamente non viene usata una sola chiave, ma una gerarchia di chiavi. Si ha quindi:

- *session key:* usata per crittografare dati per una sola sessione logica;
- *master key:* usata per cifrare le sessioni.

I problemi principali che si possono incontrare quindi sono i seguenti:

- la gerarchia delle chiavi è necessaria per reti molto vaste, ma è necessaria una sorta di garanzia sulle chiavi;
- il tempo di vita della chiave di sessione deve essere il minore possibile;
- l'uso di un sistema automatico di ditribuzione delle chiavi necessita la fiducia da parte degli utenti;
- il sistema di distribuzione è decentralizzato;
- è necessario stabilire una politica di controllo sull'uso delle chiavi.

4 Crittografia a chiave pubblica

Notazione. Oltre alla notazione specificata nella sezione 2.3, specifichiamo con PU_b e PR_b rispettivamente la chiave pubblica di B e la chiave privata di B.

4.1 Struttura del sistema

Questo tipo di crittografia elimina il problema della distribuzione delle chiavi, in quanto ogni utente ha due chiavi, una pubblica (che tutti possono vedere), e una privata (che *dovrebbe* rimanere incognita).

Ogni utente genera una coppia di chiavi, una pubblica e una privata. Quella pubblica viene inserita in un registro. Supponiamo che B voglia inviare un messaggio ad A. La procedura è la seguente:

1. B cifra il messaggio con la chiave pubblica di A;
2. A riceve il messaggio cifrato trasmesso da B;
3. A decifra il messaggio ricevuto usando la sua chiave privata.

4.2 Crittoanalisi della crittografia a chiave pubblica

Gli attacchi possibili sono i seguenti:

- *Bruteforce:* l'unica soluzione è aumentare la lunghezza della chiave, cosa che potrebbe non scalare bene con l'aumentare della dimensione, data la complessità dell'algoritmo; in pratica la crittografia a chiave pubblica viene usata solamente per la gestione delle chiavi e la firma digitale;
- *Calcolo di PR_b data PU_b*: di questo attacco non esiste prova nè controprova;
- *Probable-message attack:* supponiamo di avere un messaggio M abbastanza corto, tale che sia $C = E(PU_a, M)$, l'attaccante potrebbe calcolare tutti i $Y_i = E(PU_a, M)$ per tutti i possibili plaintext, e fermarsi quando $Y_i = C$. La soluzione a questo tipo di attacco è banale, basta appendere alcuni bit random alla fine di M, in modo tale da impedire di trovare un Y_i valido.

Il vantaggio è evidente: senza doversi scambiare le chiavi, A è certa che il messaggio non sia stato letto in precedenza, in quanto è decifrabile solo con la chiave privata che solo lei possiede. Le applicazioni sono molteplici: si va dalla firma digitale, alla cifratura/decifratura di contentuti, fino allo scambio di chiavi.

4.3 Requisiti necessari per il funzionamento

Come per la crittografia a chiave simmetrica, ci sono dei requisiti fondamentali al sistema per garantire un processo di crittografia che sia ottimale:

- deve essere facile generare la coppia di chiavi;
- deve essere facile, per il mittente A, generare $C = E(PU_b, M)$;
- deve essere facile, per il destinatario B, calcolare $M = D(PR_b, C)$;
- deve essere difficile, per un attaccante, ottenere la chiave privata da quella pubblica;
- deve essere difficile, per un attaccante, data la chiave pubblica e il ciphertext, ottenere il messaggio in chiaro.

4.4 Algoritmo RSA

Definizione 4.4.1 (One-way function) *Definiamo one-way function una funzione $f : X \to Y$ dove f è facile da calcolare $\forall x \in X$, ma è molto difficile da calcolare la sua inversa f^{-1}.*

Definizione 4.4.2 (Trapdoor one-way function) *Una trapdoor one-way function è una funzione $f_k : X \to Y$ dove, data un'informazione extra k (trapdoor) è calcolabile, $\forall y \in Im(f)$, una $x \in X$ t.c. $f_k(x) = y$.*

L'algoritmo RSA è usato in molti degli standard odierni, ma ha lo svantaggio di essere circa 1000 volte più lento di DES, oltre ad avere bisogno di chiavi abbastanza lunghe (1024 bit è relativamente sicura) ed essere vulnerabile ad alcuni tipi di attacco.

La cifratura e la decifratura iniziano da un numero noto sia ad A che a B. Il plaintext viene quindi splittato in blocchi di lunghezza pari a $\lceil log_2(n) \rceil$, in modo tale che ogni blocco rappresenti un numero M per cui $M < n$. Il ciphertext è definito come

$$C = M^e \mod n$$

mentre il plaintext è ricavabile tramite

$$M = C^d \mod n = M^{ed} \mod n$$

Le chiavi privata e pubblica sono date rispettivamente da $\{d, n\}$ e $\{e, n\}$. Perchè l'algoritmo funzioni, devono essere soddisfatti i seguenti vincoli:

- $\exists e, d, n. M^{ed} \mod n = M, \forall M < n$;
- è facile calcolare $M^e \mod n$ e $C^d \mod n$;
- è impossibile determinare d conoscendo e ed n.

Per generare una coppia di chiavi si usano i seguenti passi:

1. Si generano due numeri primi p e q (possibilmente grandi);
2. Si calcolano $n = p * q$ e $\phi = (p-1)(q-1)$;
3. Si seleziona un e, $1 < e < \phi$;
4. Si determina $d = e^{-1} \mod \phi$;
5. Si pubblica la chiave (e, n) e si mantiene privata (d, n).

La sicurezza di RSA risiede nel fatto che ricavare d data la chiave pubblica è estremamente complesso, dal momento che sarebbe necessario trovare $d = e^{-1} \mod \phi$; non si conoscono algoritmi polinomiali per fare ciò.

4.5 Distribuzione delle chiavi

Il problema risiede nella fiducia da riporre in un sistema di distribuzione delle chiavi dove le chiavi stesse non possano venire compromesse. Si usano in tal senso gli algoritmi crittografici a chiave asimmetrica.

4.5.1 Distribuzione con RSA

Lo scambio delle chiavi con RSA è abbastanza semplice: dato un m e scelta una chiave k casuale, si cifra un

$$c = (k^e \mod n, E_k(m))$$

La decifratura delle chiavi, con la chiave privata (d, n), avviene splittando il ciphertext in due blocchi separati, con

$$k = c_1^d \mod n$$
$$m = D_k(c_2)$$

L'unico problema è che se la chiave privata è compromessa, allora k può essere recuperata da un intruso dal traffico precedentemente intercettato.

4.5.2 Diffie-Hellman

Definizione 4.5.1 (Primitive root) *Una primitive root s di un numero primo p è il numero le cui potenze generano $1, \dots, p-1$.*

Definizione 4.5.2 (Logaritmo discreto) *Definiamo il logaritmo discreto di b come un valore i tale che $b = s^i \mod p$.*

Il calcolo del logaritmo discreto sembra essere infattibile, quindi è possibile strutturare un sistema crittgrafico che sfrutti questa caratteristica.

Generazione delle chiavi. La generazione delle chiavi segue le seguenti fasi:

1. I due enti si scambiano un numero primo q e una primitive root α, entrambe pubbliche;
2. A e B generano due numeri X_A e X_B, entrambi minori di q;
3. A calcola $Y_A = \alpha^{X_A} \mod q$ (analogamente B calcola Y_B);
4. A e B si scambiano i risultati;
5. A calcola $K_A = Y_B^{X_A} \mod q$, B fa l'analogo con X_B. Le due chiavi risultano essere uguali ora.

Punti di forza. Notare quali sono i punti di forza di Diffie-Hellman: la chiave è creata senza avere alcuna informazione iniziale e non è mai trasmessa (viene trasmesso solo Y)

Diffie-Hellman gode inoltre della proprietà di *perfect forward secrecy*, ossia la garanzia che le chiavi di sessione non potranno mai essere compromesse se una delle chiavi a lungo termine viene compromessa.

Debolezze. Le chiavi generate non sono autenticate, quindi sono vulnerabili ad un attacco del tipo MITM. Supponiamo infatti che nella trasmissione vengano intercettate Y_A e Y_B: in tal caso Z può calcolare le due chiavi che sarebbero calcolate da A e B, mentre A e B calcolano le relative chiavi ma utilizzando Y_Z al posto dei rispettivi Y.

Una possibile soluzione a tale problema potrebbe essere la firma digitale, ma ciò richiede l'utilizzo di una chiave condivisa.

4.6 Integrità dei messaggi

L'integrità è la proprietà che garantisce che i messaggi non sono in alcun modo stati alterati da una fonte non autorizzata. Questa proprietà viene garantita tramite funzioni di hash, ovvero funzioni che soddisfano le seguenti proprietà:

- *Compressione:* dato x come input, $h(x)$ ritorna sempre un output di lunghezza fissa;
- Sono calcolabili in tempo polinomiale.

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

- è *one-way*, ossia dato y è difficile calcolare x t.c. $h(x) = y$;
- è difficile trovare un secondo input x' tale che $h(x) = h(x')$ (*collision resistance, 2nd preimage resistance*)

Birthday attack. Un *birthday attack* sfrutta il paradosso del complanno. Supponiamo che A e B vogliano siglare un contratto, ma B vuole ingannare A facendogliene firmare uno fraudolento. B genera tanti contratti x corretti, modificandoli in modo da non cambiarne il significato, e fa altrettanto con i contratti fraudolenti y. A questo punto basta trovare due contratti x_i e y_i tali che $h(x_i) = h(y_i)$. B a questo punto fa firmare ad A x_i, ma, dato che gli hash sono uguali, può usare la firma per rendere vero anche il contratto fraudolento y_i.

4.6.1 Costruzione di una funzione hash crittografica

Uno dei metodi più semplici consiste nell'usare una tecnica di block-chaining. Si divide il messaggio in m blocchi, e si usa un algoritmo simmetrico per cifrarli uno per volta, cifrando il blocco m_i usando $E(m_{i-1})$. Tuttavia gli algoritmi moderni usano tecniche più complesse, alcuni dei quali sono:

- MD5 (hash di 128 bit, con debolezze note);
- SHA (hash di 160 bit, considerato sicuro).

4.7 Autenticazione dei messaggi

L'autenticazione dei messaggi (che implica l'integrità) garantisce la fonte del messaggio. I metodi usati per garantirla sono i MAC e le firme digitali.

4.7.1 Message Authentication Code

Un algoritmo MAC è dato da una famiglia di funzioni hash h_k (parametrizzate dalla chiave k), che devono essere *computation-resistant*, ossia, data una o più coppie $(x_i, h(x_i))$, deve essere difficile calcolare $(x, h(x))$ dato un qualsiasi $x \neq x_i$.

L'autenticità con il MAC è verificata controllando dal lato del destinatario che, se $MAC = h_k(M)$, valga $MAC' = h_k(M')$

La costruzione di un MAC avviene usando il cipher-block chaining. Dati n blocchi, si calcola

$$c_1 = E_K(m_1 \oplus 0)$$
$$c_i = E_K(c_{i-1} \oplus m_i)$$

Alla fine, il blocco c_n sarà il MAC totale.

4.7.2 Firme digitali

Lo scopo della firma digitale è dimostrare che un messaggio è stato mandato da una specifica persona, ed è fondamentale per i concetti di autenticazione e non-ripudio. Dati:

- $\mathcal{M}$ l'insieme dei messaggi firmabili;
- $\mathcal{S}$ l'insieme delle firme (stringhe di n bit);
- $S_A : \mathcal{M} \to \mathcal{S}$ è la trasformazione di firma per A (deve rimanere segreta);
- $V_A : \mathcal{M} \times \mathcal{S} \to \{true, false\}$ è la trasformazione di verifica per A, ed è pubblica.

Lo schema di firma è quindi dato da S_A e V_A.

Procedura di firma e verifica. La procedura di firma è semplice: dato $m \in \mathcal{M}$, A calcola $s = S_A(m)$ e trasmette (m, s). La procedura di verifica per A consiste nel calcolare $u = V(m, s)$. Se $u = true$ allora la firma è verificata.

Implementazione del meccanismo di firma. Lo schema per un possibile protocollo di firma reversibile (che usa crittografia simmetrica) funziona nel modo seguente:

- siano M e C il messaggio e la signature;
- siano e, d le chiavi usate per cifrare/decifrare;
- definiamo come funzione di firma $S_A = D_d$;
- definiamo quindi la verifica $V_A = \begin{cases} true & \text{if } E_e(s) = m; \\ false & \text{otherwise} \end{cases}$

Questo schema è passibile di *forgery attack*:

1. un attaccante B seleziona una $s \in S$ random e calcola $m = E_e(s)$;
2. dato che $M = S$, può submittare (m, s) come messaggio con signature;
3. la verifica ritorna *true* anche se A non ha firmato M.

La soluzione è identificare come $\mathcal{M}'$ un sottoinsieme dei messaggi firmabili, e ridefinire V_A in modo che dia true solo se $E_e(s) \in \mathcal{M}'$. In tal modo il messaggio può essere recuperato, e si ottiene un protocollo tanto sicuro quanto più piccolo è $\mathcal{M}'$.

RSA costituisce un meccanismo di firma, dal momento che la forgery è prevenuta firmando i messaggi con una struttura fissa, ossia un hash firmato, mandato col messaggio. a

5 Public Key Infrastructures

Una PKI è una infrastruttura che consente agli utenti di correlare una chiave ad un altro utente. Per registrare una chiave presso una CA (*Certification Authority*), un utente deve prima generare la sua coppia di chiavi e consegnare la chiave pubblica alla CA, la quale poi firma un certificato digitale che associa l'utente alla chiave.

Oltre al mantenimento delle chiavi, la PKI fornisce il servizio di revoca, recovery e aggiornamento delle chiavi.

I componenti di una PKI (sia aperta che chiusa) sono i seguenti:

- **Certification Authority:** crea e mantiene le directory e la CRL (Certificate Revocation List);
- **Directory:** rende disponibili la CRL e i certificati e deve identificare in modo univoco gli utenti;
- **Registration Authority:** sovrintende il processo di registrazione degli utenti e ne gestisce l'autenticazione.

5.1 Certificati

Un certificato è semplicemente un token che collega una chiave ad un'identità. Per costruire un certificato la CA firma con la sua chiave privata un messaggio contenente la rappresentazione dell'utente titolare della chiave e la sua chiave pubblica, insieme ad un timestamp.

Il problema tuttavia si ripercuote sulla CA. Come si può validare il certificato della CA? Ci sono due possibili approcci:

- costruire una gerarchia ad albero con la root trustata a priori;
- arrangiare una fiducia collettiva tra tutti.

Certificati X.509. Sono un tipo di certificato standard ora utilizzato da moltissime applicazioni (IPSEC, SSL/TLS, ecc.), e sono basati sull'uso di crittografia a chiave pubblica (RSA raccomandato).

Un normale certificato X.509 è composto dai seguenti campi:

- **Serial number:** unico tra i certificati;
- **Signature alg. identifier:** identificatore dell'algoritmo usato per firmare il certificato;
- **Issuer name:** la CA che ha firmato il certificato;
- **Periodo di validità**;
- **Subject name:** nome dell'utente a cui il certificato si riferisce (l'utente la cui chiave è certificata);
- **Subject PK info:** informazioni sulla chiave pubblica dell'utente;
- **Signature:** l'hash di tutti gli altri campi, firmato con la chiave pubblica della CA.

Se un utente vuole accedere ad un certificato e verificare la chiave di un utente procede come segue:

1. L'utente decifra la signature con la chiave pubblica della CA;
2. Usa le informazioni della signature per calcolare l'hash degli altri campi, che confronta con quello contenuto nella signature (se corrispodono il certificato è valido);
3. Controlla il periodo di validità per verificare che la chiave sia ancora valida.

Modelli fiduciari. Per stabilire la validità di un certificato esistono vari modelli di fiducia che stabiliscono in che modo gli utenti si fidano delle CA.

- **Fiducia diretta.** Ogni utente iscritto ad una CA si fida di tutti gli altri iscritti a quella CA. Tutti i certificati sono nella stessa directory.
- **Fiducia gerarchica.** Per molti utenti è più comodo avere molte CA, ognuna coprente una porzione degli utenti totali. Le CA vengono ordinate in maniera gerarchica ad albero, in cui ogni nodo non-foglia è certificato dai nodi soprastanti.

  Cross certification: se le CA si sono scambiate le corrispondenti chiavi pubbliche, allora due utenti registrati presso due CA diverse possono ottenere le rispettive chiavi attraverso una **catena di certificati**.

Key revocation. Come specificato in precedenza, la CA mantiene una CRL. Un certificato può essere revocato per svariate motivazioni:

- la chiave privata dell'utente è stata compromessa;
- l'utente non è più registrato presso quella CA;
- il certificato della CA è compromesso.

6 Realizzazione e verifica di protocolli

Prima di passare a come si realizza un protocollo, diamo alcune fondamentali definizioni.

Definizione 6.0.1 *Un protocollo consiste in un insieme di regole che determinano come avviene lo scambio di messaggi tra due utenti.*

Un protocollo di sicurezza usa meccanismi di tipo crittografico per raggiungere determinati obiettivi di sicurezza.

Per la costruzione di un protocollo è necessario stabilire una notazione di tutti gli elementi coinvolti:

- Entità che partecipano;
- Le chiavi necessarie sono:
  - cifratura: $\{M\}_K$ ($\{M\}_{K_A}$ è la cifratura con la chiave pubblica di A);
  - firma: $\{M\}_{K^{-1}}$ (la firma con la chiave privata di A è $\{M\}_{K_A^{-1}}$)
  - nel caso di chiavi simmetriche si usa $\{M\}_{K_{AB}}$
- Nonces: $NA, N1, \ldots$ numeri usati per le fasi di challenge/response;
- Timestamps T, usati per la validità delle chiavi.

Una singola comunicazione tra due entità è indicata come

$$A \to B : \{A, T_1, K_{AB}\}_{K_B}$$

È inoltre necessario fare delle assuzione sulle capacità dell'attaccante:

1. Eavesdropping completo su tutti i messaggi mandati;
2. Modifica e re-routing di tutti i messaggi;
3. L'avversario può essere uno dei principals, oppure esterno, oppure anche una combinazione dei due;
4. L'avversario è capace di ottenere il valore di tutte le precedenti chiavi di sessione (replay attack).

6.1 Protocollo di Needham - Schroeder (NSPK)

È un protocollo di scambio di chiavi, procede come segue:

1. $A \to B : \{NA, A\}_{K_B}$;
2. $B \to A : \{NA, NB\}_{K_A}$;
3. $A \to B : \{NB\}_{K_B}$

Il protocollo opera assumendo che i principals possano generare nonces, e conoscano ognuno la chiave pubblica dell'altro.

Gli obiettivi del protocollo sono:

- Autenticità dei messaggi;
- Garantire la timeliness dei messaggi (no replay attack);
- Garantire la sicurezza di specifici assets (ovvero le chiavi generate).

Il protocollo NSPK fu provato essere vulnerabile ad un attacco di tipo MITM. Fu migliorato semplicemente agguingendo alla risposta di B il nome di B. In tal modo A riceve il nome di B, si rende conto di aver scambiato il precedente messaggio con un intruso e chiude il canale.

6.2 Esempi di protocolli vulnerabili

Definiamo prima un elenco delle tipologie di attacco disponibili ad un attaccante del tipo assunto alla sezione precedente:

- Man-in-the-middle (parallel sessions): $A \leftrightarrow Z \leftrightarrow B$;
- Replay attack: riusa parti di messaggi precedenti;
- Masquerading attack: Z convince altri principals che la chiave di A sia K_Z
- Reflection attack: manda l'informazione trasmessa di ritorno al mittente;
- Oracle attack: sfrutta la disponibilità di un sistema a poter essere usato come oracolo (si/no);
- Type flaw attack: sostituisce il campo di un messaggio con un contenuto diverso (il nome per la chiave ad esempio).

L'esempio più classico è l'attacco MITM sul protocollo di Diffie-Hellman, a meno che non usi chiavi autenticate.

Type-flaw attack. Ci sono vari esempi di type-flaw attacks. Gli esempi più classici sono il protocollo Otway Rees (Authenticated server based key distribution), e l'Andrew Secure RPC.

Nel caso Otway-Rees, il protocollo è così composto:

$$
\begin{aligned}
&M_1.A \to B : I, A, B, \{N_A, I, A, B\}_{K_{AS}} \\
&M_2.B \to S : I, A, B, \{N_A, I, A, B\}_{K_{AS}}, \{N_B, I, A, B\}_{K_{BS}} \\
&M_3.S \to B : I, \{N_A, K_{AB}\}_{K_{AS}}, \{N_B, K_{AB}\}_{K_{BS}} \\
&M_4.B \to A : I, \{N_A, K_{AB}\}_{K_{AS}}
\end{aligned}
$$

Le chiavi del server sono note e I indica una run del protocollo. La vulnerabilità risiede nel fatto che, supponendo $|\{I, A, B\}| = |\{K_{AB}\}|$, un attaccante può inoltrare un messaggio del tipo 1 come messaggio del tipo 4. In tal modo A vede il suo nonce e accetta (I, A, B) come chiave di sessione.

Per questo protocollo si prostpetta un secondo scenario di attacco, ovvero quello in cui l'attaccante fa da server, e inoltre a B, invece della chiave, la tripla (I, A, B). In tal modo B e A usano (I, A, B) come chiave e la secrecy non vale più.

Nel caso Andrew-Secure RPC, il protocollo funziona così:

$$
\begin{aligned}
&M_1.A \to B : A, \{N_A\}_{K_{AB}} \\
&M_2.B \to A : \{N_A + 1, N_B\}_{K_{AB}} \\
&M_3.A \to B : \{N_B + 1\}_{K_{AB}} \\
&M_4.B \to A : \{K'_{AB}, N'_B\}_{K_{AB}}
\end{aligned}
$$

K'_{AB} è la chiave di sessione e N'_B è il nonce per la sessione seguente. L'attaccante può muoversi in modo che, in M_4, la chiave scambiata sia $N_A + 1$, previa intercettazione di M_2. Ciò nonostance, la segretezza dei messaggi non è violata.

Protocollo Denning - Sacco. Il protocollo si svolge in 3 passi:

$$
\begin{aligned}
&A \to S : A, B \\
&S \to A : C_A, C_B \\
&A \to B : C_A, C_B, \{\{T_A, K_{AB}\}_{K_A^{-1}}\}_{K_B}
\end{aligned}
$$

dove:

- S è un server che funge da CA;

- K_{AB} chiave di sessione segreta;
- K_B chiave pubblica di B;
- K_A^{-1} chiave privata di A, usata per firma;
- C_A, C_B sono i certificati per A e B;
- T_A timestamp generato da A.

Il punto di forza di questo protocollo risiede nel fatto che B è sicuro che la chiave pubblica sia di A in quanto è firmata con la sua chiave privata, il tutto cifrato con la chiave pubblica di B, il che indica che il messaggio è per B. Come extra, la chiave di A è provata da C_A.

Questo protocollo è vulnerabile ad un MITM, ma è risolvibile aggiungendo i nomi di A e B insieme al timestamp e la chiave. In tal modo è facile verificare se qualcuno usa la propria chiave per modificare la cifratura della firma.

Parallel sessions attack. Consideriamo un semplice (ed estramamente lazy) protocollo di autenticazione:

$$M_1.A \to B : \{N_A\}_{K_{AB}}$$
$$M_2.B \to A : \{N_A + 1\}_{K_{AB}}$$

Il protocollo è vulnerabile ad un attacco di tipo parallel sessions (con oracolo). L'attacco è banale:

$$M_1.A \to I : \{N_A\}_{K_{AB}}$$
$$M_2.I \to A : \{N_A\}_{K_{AB}}$$
$$M_3.A \to I : \{N_A + 1\}_{K_{AB}}$$
$$M_4.I \to A : \{N_A + 1\}_{K_{AB}}$$

L'unica soluzione a questo attacco è, come nei casi precedenti, aggiungere il nome di A nel messaggio cifrato.

Replay attack. Consideriamo il protocollo Needham-Schroeder per chiavi condivise:

$$M_1.A \to S : A, B, N_1$$
$$M_2.S \to A : \{N_1, B, K_{AB}, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$$
$$M_3.A \to B : \{K_{AB}, A\}_{K_{BS}}$$
$$M_4.B \to A : \{N_2\}_{K_{AB}}$$
$$M_5.A \to B : \{N_2 - 1\}_{K_{AB}}$$

Tale protocollo è passabile di replay attack, in quanto un qualsiasi attaccante Z, supponendo che abbia recuperato la chiave di una precedente run del protocollo (K'_{AB}), potrebbe agire come segue:

$$M_3.Z \to B : \{K'_{AB}, A\}_{K_{BS}}$$
$$M_4.B \to A : \{N_2\}_{K'_{AB}}$$
$$M_5.A \to B : \{N_2 - 1\}_{K'_{AB}}$$

In tal modo verrebbe compromessa la segretezza dei messaggi. Un possibile fix è quello di aggiungere dei timestamp da usare come timeout, oppure aggiungere un handshake extra all'inizio del protocollo.

7 Kerberos

Kerberos è un protocollo di autenticazione in ambienti distribuiti. È costruito su alcuni principi di base:

- Sicurezza (no eavesdropping);
- Affidabilità;
- Trasparenza: l'utente usa un password singola per l'accesso alla rete, ignaro dei protocolli sottostanti (SSO);
- Scalabilità: il sistema deve essere in grado di supportare molti utenti/servizi alla volta.

7.1 Architettura e funzionamento

L'architettura Kerberos è strutturata come segue:

- KAS (*Kerberos Authentication Server*) per l'autenticazione;
- TGS (*Ticket Granting Server*) per l'autorizzazione;
- Servizi di controllo dei ticket.

Supponiamo ora che un utente voglia accedere ad un servizio. L'architettura funziona come segue:

1. L'utente richiede il servizio, richiedendo al KAS un ticket-granting ticket;
2. Il KAS verifica l'accesso, genera il ticket e crea la chiave di sessione. Il tutto viene cifrato con una chiave calcolata usando la psw di accesso.
3. La workstation dell'utente chiede la password, che usa per decifrare il messaggio in arrivo, poi manda l'authenticator (username, indirizzo di rete e timestamp attuale) al TGS;
4. il TGS decifra il tutto, verifica la richiesta, genera il ticket per il server con il servizio richiesto e manda il tutto alla workstation;
5. La workstation manda il ticket per il server e l'authenticator al server stesso;
6. Il server verifica che ticket e authenticator matchino, quindi da accesso al servizio. Se è richiesta autenticazione mutuale, il server manda indietro il suo authenticator.

È importante sottolineare che la fase 1 e 2 (**autenticazione**) sono svolte ad ogni sessione di login, i messaggi 3 e 4 (**autorizzazione**) sono svolti per ogni tipo di servizio mentre i messaggi 5 e 6 (**accesso al servizio**) sono scambiati ad ogni accesso al servizio.

Autenticazione. La fase di autenticazione procede come:

$$1.A \to KAS : A, TGS$$
$$2.KAS \to A : \{K_{A,TGS}, TGS, \mathcal{T}_1, \{A, TGS, K_{A,TGS}, \mathcal{T}_1\}_{K_{KAS,TGS}}\}_{K_{AS}}$$

La sezione $\{A, TGS, K_{A,TGS}, \mathcal{T}_1\}_{K_{KAS,TGS}}$ è l'*AuthTicket*. $K_{A,TGS}$ ha un lifetime di alcune ore.

Autorizzazione. La fase di autorizzazione contiene i messaggi 3 e 4, ovvero

$$3.A \to TGS : \textit{AuthTicket}, \{A, T_2\}_{K_{A,TGS}}, B$$
$$4.TGS \to A : \{K_{AB}, B, T_3, \{A, B, K_{AB}, T_3\}_{K_{B,TGS}}\}_{K_{A,TGS}}$$

La sezione del messaggio 3 $\{A, T_2\}_{K_{A,TGS}}$ è l'*autenticator*, che ha un periodo di validità molto ridotto (secondi). Lo scopo dell'autenticator è quello di scongiurare eventuali replay attacks. Il TGS fornisce la chiave K_{AB} con lifetime di alcuni minuti, e un *ServTicket* ($\{A, B, K_{AB}, T_3\}_{K_{B,TGS}}$)

Service phase. La fase di accesso al servizio procede con i messaggi 5 e 6, ovvero:

$$5. A \rightarrow B : \textit{ServTicket, Authenticator}$$
$$6. B \rightarrow A : \{T_4 + 1\}_{K_{AB}}$$

Dopo la risposta del server, l'utente è autenticato e può iniziare ad usare il servizio.

7.2 Scalabilità

Definiamo *realm* lo spazio definito da un server Kerberos. Il server salva username e password solo per gli utenti del suo realm. Tuttavia, una grande rete potrebbe essere divisa in più realms. Per questo motivo Kerberos è dotato anche di protocolli inter-realm, dove i server si registrano l'un l'altro e, nel caso un utente dovesse accedere ad un servizio posizionato in un altro realm, il KAS fornisce il ticket per accedere al TGS dell'altro realm.

Kerberos 4 è soggetto a particolari limitazioni, quali il possibile flooding del KAS da parte di un attaccante, e la pesantezza della doppia crittografia, rimossa in Kerberos 5.

Tutto il meccanismo inotre si basa sulla sincronizzazione dei timestamp, pertanto, se un host dovesse essere compromesso, per l'attaccante potrebbe essere possibile modificare i timestamps e attuare un replay attack.

8 SSL/TLS

TLS è un protocollo nato dall'evoluzione di SSL, che garantisce una cifratura end-to-end anche in presenza di una rete compromessa. TLS è composto da un **Record Protocol** che protegge i dati scambiati tra client e server, e l'**Handshake protocol**, reponsabile della fase di handshake tra le parti.

8.1 SSL Handshake

L'handshake è composto di 4 fasi:

1. Decisione dei protocolli crittografici da adottare;
2. Scambio di chiavi del server;
3. Scambio di chiavi del client;
4. Fine, connessione online.

Decisione dei protocolli. I messaggi scambiati in questa fase sono 2:

1. `client_hello`: il client invia al server un messaggio $\{A, N_A, S_{id}, P_A\}$, contenente il nome di A, un nonce, il session id e le preferenze sugli algoritmi per lo scambio delle chiavi e di compressione;
2. `server_hello`: lo stesso messaggio, ma con gli algoritmi e il nonce del server.

Scambio chiave del server. Il sever manda al client un certificato X.509v3 (`server_certificate`) $cert(S, K_S)$. Notare che in realtà il protocollo prevede che si possa anche negoziare l'algoritmo per lo scambio delle chiavi e anche richiedere un certificato per il client (supponendo che il client abbia un propria chiave pubblica).

Scambio chiavi del client. Il client risponde con:

1. `client_certificate`: il proprio certificato (se può);
2. `client_key_exchange`: *pre-master secret*, PMS, cifrato con la chiave pubblica del server (arrivata tramite il certificato della fase 2). Il PMS sarà usato per calcolare il *master secret* M, che ha utilizzi più avanti;
3. `certificate_verify`: la risposta al controllo di validità del certificato.

Fine della comunicazione. Client e server si scambiano due messaggi `finish`, cifrati con chiavi simmetriche generate dai nonce e M. Il messaggio di finish è un hash dei precedenti messaggi, garantendo integrità di tutti i messaggi precedenti.

8.2 Attacchi ad SSL

Version rollbcack attack. L'attaccante trickava il server ad usare un versione più vecchia di SSL (già violata), per poter accedere al contentuto del protocollo (analogamente, è possibile trickare il client ad usare protocolli di cifratura deboli).

Per questo motivo, il protocollo fu aggiornato includendo nei messaggi cifrati anche la versione del protocollo, affinchè nessun attaccante tra client e server potesse intercettare i messaggi e reinoltrarli con la versione più vecchia di SSL (si parla di *chosen-protocol*).

Phishing - vulnerabilità UI. Un altro problema è rappresentato dagli utenti, che, anche se sotto attacco, non si prendono la briga di controllare con quale azienda stanno scambiando dati (controllando il certificato in uso), oppure sono soggetti a phishing.

SSLstrip. In tal caso, l'attaccante si trova tra client e server e, supponendo sia in grado di intercettare le pagine web che client e server stanno scambiando, può modificare, rimpiazzando il protocollo HTTPS con HTTP, ottenendo così informazioni sensibili senza che client e server ne vengano a conoscenza.

Vulnerabilità nello storage delle chiavi. In questo caso, la vulnerabilità non risiede nel client nè nel protocollo in sè, quanto nel modo in cui le chiavi sono immagazzinate nel server (molto spesso non cifrate), per non dover inserire la passphrase ad ogni boot.

9 IPSec

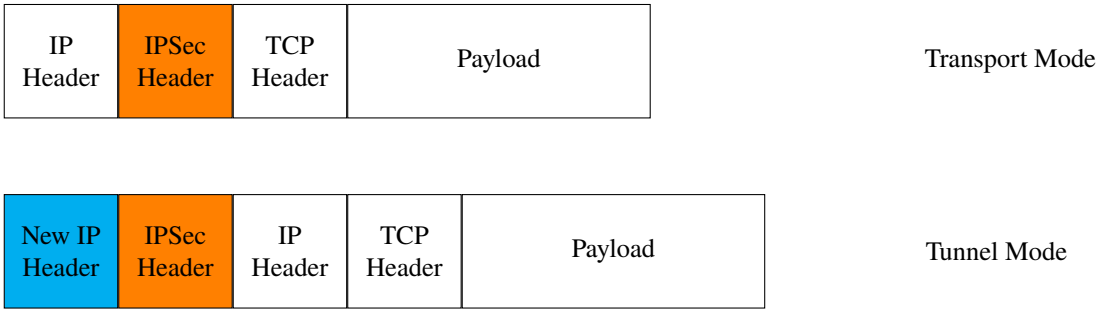
IPSec è un protocollo che implementa un canale sicuro di trasmissione per le applicazioni, cifrando e autenticando tutto il traffico in uscita da un host. IPSec di per sè garantisce:

- **Confidenzialità**, cifrando i dati;
- **Integrità**, in quanto i router alla fine del tunnel calcolano un hash o un checksum dei dati in arrivo;
- **Autenticazione**, attraverso un meccanismo di firme e certificati.

Definiamo *Security Association* una one-way relationship tra due entità, e definisce delle politiche sui servizi di sicurezza, come ad esempio il protocollo crittografico usato, l'hash per l'autenticazione, ecc.
IPSec in sè è dotato di un insieme di protocolli all'interno della specifica:

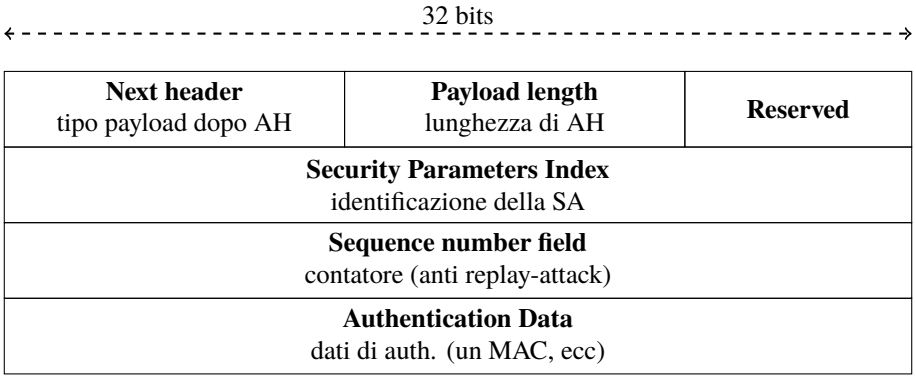
- **Authentication Header (AH)** per l'integrità e l'autenticità dei pacchetti;
- **Encapsuating Security Payload (ESP)** per la protezione della confidenzialità (integrità in modo opzionale);
- **Key Management (IKE)** per la gestione delle chiavi.

IPSec può operare in due modalità: una **tunnel mode** in cui ogni pacchetto viene cifrato e diventa parte di un pacchetto più grande (site-to-site VPN), oppure la **transport mode**, che consiste nell'inserire l'header IPSec all'interno di un normale pacchetto IP, senza creare nuovi pacchetti (remote access VPN).



9.1 Authentication Header

L'authentication header sta tra l'header IP e quello TCP, e fornisce al destinatario informazioni per identificare la SA. L'header è strutturato come segue:



L'AH in transport mode è inserito tra l'header IP e il payload, e il MAC viene fatto di tutto il pacchetto. In tunnel mode tutto il pacchetto è autenticato, il nuovo header IP, così come l'AH sono inseriti per ultimi. L'AH fornisce quindi un canale autenticato end-to-end.

9.2 Encapsulating Security Payload

L'header ESP specifica solo il protocollo di cifratura e, in maniera opzionale, per l'autenticazione. In transport mode (end-to-end encryption tra host che supportano IPSec) viene cifrato solo il payload del pacchetto (l'header IP rimane invariato), mentre in tunnel mode (VPN setup) tutto il pacchetto viene cifrato, ad esclusione del nuovo header IP.

Riassumiamo i servizi offerti da IPSec come segue:

	AH	**ESP (encr. only)**	**ESP w/auth**
Access control	✓	✓	✓
Connectionless integrity	✓		✓
Data origin auth.	✓		✓
Replayed packets rejec.	✓	✓	✓
Confidentiality		✓	✓
Limited traf. flow conf.		✓	✓

9.3 IKE - Internet Key Exchange

IKE è un protocollo che, a dispetto del nome, può stabilire anche delle SA. È prodotto dall'evoluzione di vecchi protocolli come ISAKMP (framework per lo scambio di chiavi e gestione delle SA), e Oakley (joint key generation), entrambi basati su Diffie-Hellman.

Il punto con Diffie-Hellman risiede nella PFS (*Perfect Forward Secrecy*), ovvero il concetto secondo il quale un attaccante che registra un'intera conversazione cifrata non la può decifrare anche se possiede le chiavi di entrambi gli utenti. La chiave sta nel generare una chiave di sessione temporanea, non derivabile dalle informazioni salvate da ogni nodo.

IKE procede in 2 fasi:

1. Le due entità che comunicano negoziano una SA, accordandosi sui cifrari e le funzioni di hash da usare nella fase 2;
2. l'SA della fase 1 è usata per creare delle SA figlie, con lo scopo di cifrare e autenticare le future comunicazioni (non spiegato qua).

Fase 1. La fase uno può eseguire in due modalità differenti, entrambi producenti una SA: una **main mode**, molto più flessbile, con meccanismi di protezione delle identità, e una **aggressive mode**, che scambia la metà dei messaggi, ma non prevede alcun meccanismo di protezione dell'identità. Ognuna di queste modalità ammette 4 varianti, dipendenti dal metodo di autenticazione.

Main mode. La main mode è composta dei seguenti messaggi:

1. $I \to R : C_I, ISA_I$: l'intiator manda al responder l'SA con gli algoritmi che supporta;
2. $R \to I : C_I, C_R, ISA_R$ il responder risponde con gli algoritmi scelti;
3. $I \to R : C_I, C_R, X, N_I$ l'intiator manda al responder il suo X di DH e un suo nonce;
4. $I \to R : C_I, C_R, Y, N_R$ il responder risponde con il suo Y, calcolato da g e p inviati insieme ad X, e il suo nonce;
5. $I \to R : C_I, C_R, \{ID_I, AUTH_I\}_K$;
6. $I \to R : C_I, C_R, \{ID_R, AUTH_R\}_K$.

I messaggi 5 e 6 sono cifrati con la chiave K, generata precedentemente con DH. Gli ID sono identificatori di host, mentre $AUTH$ sono dati di autenticazione, dipendenti dalle varianti utilizzate.

Quick mode. La quick mode è un condensato della main mode in 3 messaggi, senza l'autenticazione dell'identità dei due interlocutori:

1. $I \rightarrow R$: autenticazione, materiale per il calcolo della chiave e proposta della SA;
2. $R \rightarrow I$: SA accettata, materiale di risposta della chiave;
3. $I \rightarrow R$: hash totale per il controllo di integrità.