Politecnico di Torino
Laurea Magistrale in Ingegneria Informatica

appunti di

Sicurezza dei sistemi informatici

Autori principali: Nicola Gallo, Loris Gabriele, Luca Ghio, Lorenzo Liotino, Muhammad Ali
Docenti: Antonio Lioy
Anno accademico: 2014/2015
Versione: 1.0.0.0
Data: 28 febbraio 2015

Ringraziamenti

Oltre agli autori precedentemente citati, quest'opera può includere contributi da opere correlate su WikiAppunti e su Wikibooks, perciò grazie anche a tutti gli utenti che hanno apportato contributi agli appunti *Sicurezza dei sistemi informatici* e al libro *Sicurezza dei sistemi informatici*.

Informazioni su quest'opera

Quest'opera è pubblicata gratuitamente. Puoi scaricare l'ultima versione del documento PDF, insieme al codice sorgente LaTeX, da qui: http://lucaghio.webege.com/redirs/a

Quest'opera non è stata controllata in alcun modo dai professori e quindi potrebbe contenere degli errori. Se ne trovi uno, sei invitato a correggerlo direttamente tu stesso realizzando un commit nel repository Git pubblico o modificando gli appunti *Sicurezza dei sistemi informatici* su WikiAppunti, oppure alternativamente puoi contattare gli autori principali inviando un messaggio di posta elettronica a nicola.gallo90@hotmail.it o a artghio@tiscali.it.

Licenza

Immagini

- capitolo 10: © Antonio Lioy - Politecnico di Torino (2009-2014)
- capitolo 11: © Antonio Lioy - Politecnico di Torino (2013-2015)

Indice

Capitolo 1

Introduzione alla sicurezza delle reti e dei sistemi informativi

sicurezza informatica l'insieme dei prodotti, dei servizi, delle regole organizzative e dei comportamenti individuali che proteggono i sistemi informatici di un'azienda

- prodotti: l'azienda deve comprare i prodotti giusti per la sicurezza;
- servizi: i prodotti offrono dei servizi di sicurezza;
- regole organizzative: i prodotti e i servizi devono essere usati in un certo modo;
- comportamenti individuali: il comportamento di un singolo individuo può rovinare la sicurezza di tutta l'azienda.

La sicurezza informatica ha il compito di garantire la **C.I.A.** = "Confidentiality, Integrity, Availability".

L'obiettivo è custodire le informazioni con la stessa professionalità ed attenzione con cui ci si prende cura di gioielli o certificati azionari depositati nel caveau:

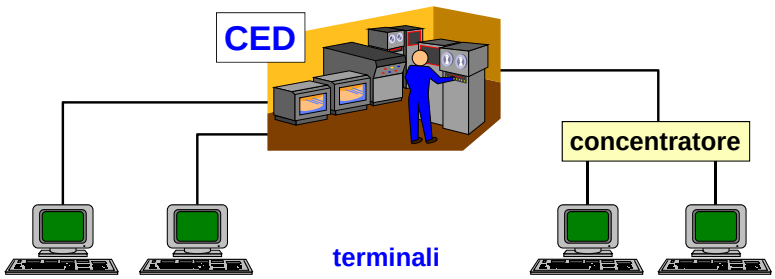
- il sistema informatico è la cassaforte delle nostre informazioni più preziose;
- la sicurezza informatica è l'equivalente delle serrature, combinazioni e chiavi che servono a proteggerla.

1.1 Perché è necessaria la sicurezza?

Il problema della sicurezza delle reti e dei sistemi informativi è sempre esistito fin dagli anni '60, ma oggi sta diventando sempre più importante a causa di tre principali fattori:

- tecnologico: mentre in passato un attaccante doveva recarsi fisicamente nel luogo dove si trovava il sistema che intendeva danneggiare, oggi un attacco a un sistema informatico può arrivare da ogni parte nel mondo attraverso la rete;
- economico: gli attacchi informatici possono causare danni economici significativi, anche indiretti: ad esempio, un attacco al sito di una banca può essere percepito dai clienti come un rischio per i loro conti in banca;
- strategico: la protezione dei sistemi informatici che controllano i servizi pubblici (acquedotti, ferrovie, centrali atomiche, ecc.) è di importanza nazionale, poiché molte vite dipendono dal loro corretto funzionamento.

1.1.1 Paradigma tradizionale



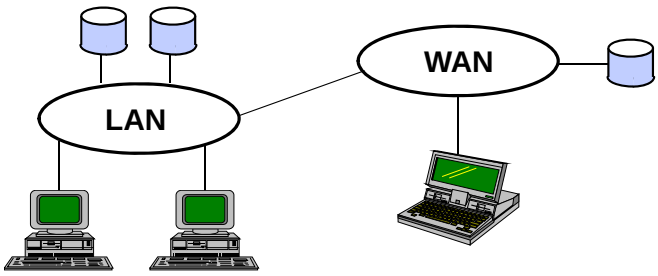
Secondo il vecchio paradigma tradizionale, tutti i dati e tutta la capacità di elaborazione sono inclusi in un singolo grande computer, detto **mainframe**, posto in una stanza chiamata Centro Elaborazione Dati (CED), e l'accesso al mainframe è centralizzato:

- accesso diretto: inizialmente era necessario inserire delle schede perforate in un apposito lettore, che era sotto il controllo diretto dell'amministratore di sistema;
- accesso remoto: successivamente l'accesso a distanza fu reso possibile dalla comparsa dei **terminali**, postazioni "stupide" composte da schermo e tastiera senza alcuna capacità di elaborazione e di memorizzazione (solo input/output), che erano collegati direttamente alla sede centrale tramite **linee dedicate** end-to-end (più terminali di una filiale potevano essere aggregati in multiplexing in una singola linea dedicata tramite un concentratore).

Un sistema di questo tipo deve essere protetto:

- a livello fisico: l'infrastruttura fisica stessa è esposta ad attacchi:
  - sede centrale: il CED deve essere un luogo sicuro per impedire a un attaccante l'accesso fisico alla macchina;
  - linee dedicate: è anche necessario proteggere i cavi di collegamento;
- a livello logico: il sistema operativo e le applicazioni devono includere dei sistemi di sicurezza di base:
  - autenticazione: è necessario implementare un sistema per l'autenticazione degli utenti (ad es. nome utente e password), in modo che il mainframe venga usato solo dalle persone autorizzate;
  - autorizzazione: è necessario definire le operazioni che ad ogni singolo utente è permesso compiere una volta autenticato.

1.1.2 Evoluzione dello scenario



Oggi la situazione è molto cambiata:

- i dati e l'elaborazione sono distribuiti ⇒ è necessario proteggere più elaboratori, non solamente uno centralizzato;
- i terminali sono "intelligenti": sono dotati di capacità di elaborazione e di memorizzazione autonome;
- le comunicazioni sono broadcast attraverso **linee condivise** ⇒ poiché i dati viaggiano su un canale condiviso tra tutti i nodi, un utente malintenzionato potrebbe sniffarli facilmente;
- i sistemi stanno diventando sempre più complessi a causa dello sviluppo di nuovi paradigmi applicativi (come il Web, il P2P, gli SMS) ⇒ è sempre più difficile prevedere tutti i tipi di attacco possibili.

L'evoluzione della tecnologia spesso non è seguita da cambiamenti nei sistemi di sicurezza:

- le **reti** sono insicure:
  - le comunicazioni avvengono per lo più in chiaro: si assume che nessun altro voglia leggere i dati trasmessi;
  - le reti locali (LAN) funzionano logicamente in unicast, ma fisicamente in broadcast: si assume che chi non è il destinatario ignori i dati ricevuti;
  - nelle reti geografiche (WAN) le connessioni avvengono tramite router di terzi: si assume che i dati in transito non siano letti e manipolati da chi controlla il router;
- l'**autenticazione** è trascurata:
  - l'autenticazione degli utenti è debole, basandosi semplicemente su nome utente e password: si assume che chi ha inserito la password ne sia veramente il titolare;
  - non c'è autenticazione dei server: l'utente normalmente si fida del server;
- il **software** contiene molti bug, alcuni dei quali costituiscono delle vulnerabilità di sicurezza sfruttabili per compiere degli attacchi (ad es. attacchi DoS):
  - mancanza del controllo del buffer overflow (ad es. funzione `gets()`): se vengono inseriti più dati di quanti possono stare nel buffer, i dati in eccesso sovrascrivono altre zone di memoria ⇒ l'attaccante può eseguire del codice arbitrario iniettato tramite un input opportunamente manipolato;
  - memorizzazione di informazioni sensibili nei cookie: i cookie possono essere letti da terzi sia durante il transito in rete sia quando sono memorizzati sul client;
  - memorizzazione delle password in chiaro in un database: possono essere lette da terzi ⇒ l'informazione memorizzata (ad es. il risultato di una funzione di hash) deve limitarsi a permettere di verificare se la password che l'utente ha inserito è corretta;
  - implementazione di un sistema di protezione inventato: la protezione rischia di essere inadeguata a causa di errori di progettazione.

1.1.3 Le cause profonde della insicurezza[1]

"La tecnologia di attacco si sta sviluppando in un ambiente open-source e si sta evolvendo rapidamente"
Poiché molti strumenti di attacco vengono rilasciati in licenza open source, sono molto facili da trovare e da migliorare ⇒ si evolvono molto più rapidamente rispetto agli strumenti di difesa, che spesso sono prodotti commerciali.

[1] Analisi del documento *Consensus Roadmap for Defeating Distributed Denial of Service Attacks* (2000).

"le strategie difensive sono reazionarie"
Le strategie di difesa sono reazionarie: si tende a reagire solo dopo aver ricevuto un attacco ⇒ i difensori sono sempre in ritardo rispetto agli attaccanti. Per esempio, un aggiornamento dell'antivirus viene rilasciato dopo che un nuovo virus è nato e ha cominciato a infettare qualche computer.

"ci sono decine di migliaia – forse anche milioni – di sistemi con sicurezza debole connessi ad Internet"
Non è sufficiente che il sistema su cui si sta lavorando sia protetto per essere al sicuro da eventuali attacchi, ma anche tutti gli altri sistemi interconnessi all'interno della rete devono essere sicuri, altrimenti il nodo più debole può essere usato come punto di partenza per un attacco all'intero sistema.

"L'esplosione nell'utilizzo di Internet sta mettendo a dura prova il nostro scarso talento tecnico. Il livello di competenza tecnica medio degli amministratori di sistema è diminuito drammaticamente negli ultimi 5 anni"
Le interfacce grafiche permettono di eseguire delle operazioni automatizzate che non sono sufficienti quando è necessaria un'analisi approfondita del problema.

"Software sempre più complesso è scritto da programmatori che non hanno alcuna formazione nella scrittura di codice sicuro"
La sicurezza non deve essere l'ultimo componente a cui pensare, ma deve essere considerata già in fase di progettazione: è importante sviluppare del codice che preveda i casi di input errati da parte dell'utente, altrimenti un attaccante potrebbe inviare degli input non previsti per mandare in crash il sistema. Il numero di bug cresce esponenzialmente con il numero di righe di codice (LOC) ⇒ conviene suddividere il programma in più componenti e testarle separatamente per ridurre il numero di bug.

"L'evoluzione della tecnologia di attacco e la distribuzione degli strumenti di attacco travalicano la geografia e i confini nazionali"
L'origine dell'attacco può essere impossibile da individuare (ad es. l'attaccante ha fatto in modo che dei computer intermedi lo separino dal bersaglio).

"La difficoltà di indagine penale del cybercrimine unitamente al diritto internazionale significano che [...] il perseguimento dei reati informatici è improbabile"
L'attaccante può non essere perseguibile se ha agito in uno Stato dove la legislazione per i reati informatici non è ben definita. Inoltre l'alto numero di attacchi scoraggia il perseguimento di tutti i colpevoli di tali reati.

1.2 Proprietà di sicurezza

Quando si progetta un sistema di sicurezza, è necessario definirne alcune proprietà astratte:

- **autenticazione**: chi sei? (sezione 1.2.1)
- **non ripudio**: sei tu l'autore? (sezione 1.2.2)
- **autorizzazione**: puoi farlo? (sezione 1.2.3)
- **riservatezza**: qualcuno ha accesso ad informazioni riservate? (sezione 1.2.4)
- **integrità**: i dati sono stati manipolati? (sezione 1.2.5)
- **disponibilità**: il sistema è funzionante per fornire il servizio a fronte di eventi imprevedibili (ad es. attacchi DoS)?

- **tracciabilità**: dopo che si è verificato un attacco, è possibile risalire a chi è l'attaccante?

> **disponibilità** la proprietà di essere accessibile e utilizzabile su richiesta di un'entità autorizzata
>
> **tracciabilità** la proprietà di un sistema o risorsa di sistema che garantisce che le azioni di un'entità di sistema possono essere tracciate univocamente a quell'entità, che può quindi essere ritenuta responsabile delle sue azioni

Possono servire tutte queste proprietà o solamente alcune di esse, a seconda del tipo di sistema di sicurezza da implementare.

1.2.1 Autenticazione

> **autenticazione** il processo di verifica di una dichiarazione che un'entità di sistema o una risorsa di sistema ha un certo valore di attributo
>
> **autenticazione dell'entità pari** la corroborazione che un'entità pari in un'associazione è quella dichiarata
>
> **autenticazione dell'origine dati** la corroborazione che la sorgente dei dati ricevuti è come dichiarato

Un sistema di sicurezza offre la proprietà di **autenticazione** se consente di verificare una prova che ha il fine di attestare la veridicità di un attributo di dati o entità:

- **autenticazione della controparte**: chi sta dall'altra parte del canale di comunicazione fornisce una prova che attesta la sua identità:
  - semplice: una delle due controparti dimostra la sua identità all'altra e non viceversa (ad es. l'utente fornisce nome utente e password al server);
  - mutua: anche l'altra controparte dimostra la sua identità (ad es. un sito Web fornisce un certificato digitale all'utente);
- **autenticazione** intrinseca[2] **dei dati**: i dati stessi contengono una prova che attesta la veridicità di un loro attributo:
  - autenticazione dell'origine dati: i dati contengono una prova che attesta la veridicità della sorgente (ad es. un messaggio di posta elettronica contiene la firma digitale del mittente).

L'autenticazione è un concetto diverso dall'**identificazione**:

- autenticazione: è il risultato di una procedura elettronica (ad es. inserimento di nome utente e password);
- identificazione: significa avere la certezza che le credenziali di accesso siano state effettivamente inserite dal loro titolare e non da persone terze.

L'autenticazione dovrebbe essere la proprietà più importante in un sistema di sicurezza: se l'autenticazione non è solida, tutte le misure di sicurezza ai livelli soprastanti non hanno senso. Tenere traccia delle operazioni che sono state svolte (log) ha senso dal punto di vista della sicurezza solo se si ha la certezza di chi ha effettuato quelle operazioni (autenticazione). Tuttavia, nella realtà sistemi di grandissimo valore economico sono costruiti su una base altamente instabile, per esempio affidandosi a un'autenticazione basata semplicemente su nome utente e password.

[2] Non si può fare un'autenticazione in tempo reale quando non c'è un canale di comunicazione diretto.

(a) ideale *(b) reale*

Figura 1.1: Piramide della sicurezza.

1.2.2 Non ripudio

ripudio negazione da parte di una delle entità coinvolte in una comunicazione di aver partecipato in tutta la comunicazione o in parte di essa

Un sistema di sicurezza offre la proprietà di **non ripudio** se consente di creare una prova formale, usabile in tribunale, che dimostra in modo innegabile l'autore dei dati.

Una persona o una società potrebbe avere interesse a negare di essere l'autore di dati legati ad azioni illegali (ad es. negare di avere inviato un messaggio di posta elettronica minatorio).

Per ottenere questa proprietà, molti aspetti sono da considerare: autenticazione del mittente, integrità dei dati, identificazione del mittente, ecc.

1.2.3 Autorizzazione

autorizzazione un'approvazione che è concessa a un'entità di sistema per accedere a una risorsa di sistema

controllo accessi protezione di risorse di sistema da accessi non autorizzati

Un sistema di sicurezza offre la proprietà di **autorizzazione** se consente di definire quali operazioni un utente ha il permesso di effettuare all'interno del sistema.

L'autorizzazione spesso coincide con il **controllo accessi**: quando un utente cerca di accedere a una risorsa per svolgere una operazione, il modulo di autorizzazione ha il compito di verificare se ha i necessari permessi.

1.2.4 Riservatezza

riservatezza il diritto degli individui di controllare o di influenzare quali informazioni relative ad essi possono essere raccolte e memorizzate e da chi e a chi possono essere divulgate quelle informazioni

confidenzialità la proprietà che le informazioni non sono rese disponibili né divulgate a individui, entità o processi non autorizzati

Un sistema di sicurezza offre la proprietà di **riservatezza** (o confidenzialità) se consente di garantire che nessuno abbia accesso ad informazioni confidenziali senza essere autorizzato:

- riservatezza delle comunicazioni: i dati che attraversano un canale di comunicazione possono essere intercettati (ad es. il traffico di rete può essere sniffato se non è criptato);
- riservatezza dei dati: i dati possono essere rubati dai supporti fisici su cui sono memorizzati (ad es. se i file non sono memorizzati criptati su un disco fisso);
- riservatezza delle azioni: le azioni compiute possono essere registrate (ad es. un ISP può tracciare i siti Web visitati dagli utenti);
- riservatezza della posizione: la posizione può essere registrata (ad es. un ladro può compiere un furto quando l'inquilino è fuori casa).

1.2.5 Integrità

integrità dei dati la proprietà che i dati non sono stati alterati o distrutti in una maniera non autorizzata

replay attack un attacco in cui una trasmissione dati valida viene ripetuta in modo malevolo o fraudolento, o dall'originatore o da un terzo che intercetta i dati e li ritrasmette, eventualmente come parte di un attacco masquerade

Un sistema di sicurezza offre la proprietà di **integrità** se consente di verificare se i dati sono stati manipolati:

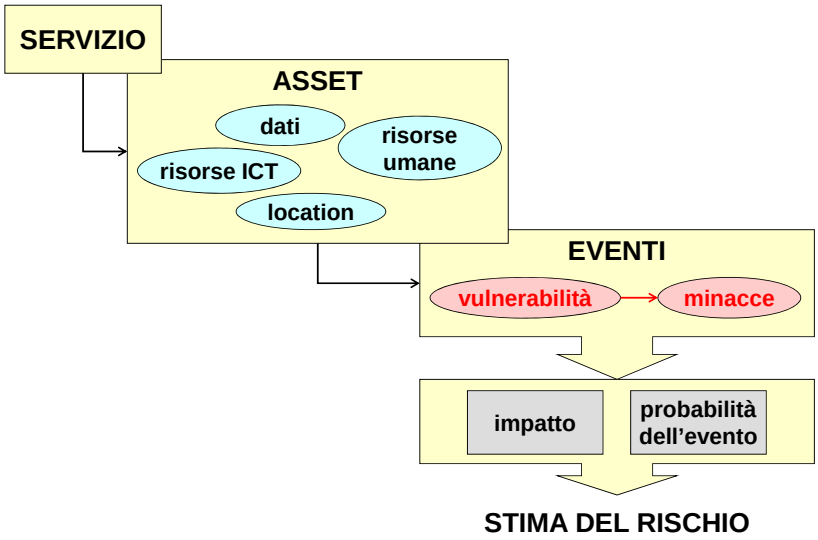
- modifica: i dati trasmessi o memorizzati possono venire modificati;
- cancellazione: i dati in transito possono venire bloccati per impedire che arrivino a destinazione ⇒ il trasmettitore può richiedere che il ricevitore mandi un acknowledgment per confermare la ricezione, ma l'acknowledgment può essere falsificato;
- ripetizione: i dati in transito, anche criptati, possono essere copiati in modo da riuscire a ottenere l'effetto (ad es. pagamento) ogni volta che viene trasmessa una copia dei dati (**replay attack**) ⇒ occorre inserire dei numeri identificativi, non modificabili dagli attaccanti, per rilevare repliche dei pacchetti.

1.3 Analisi e gestione della sicurezza

Per garantire la sicurezza di un sistema informatico, è necessario:

- analisi della sicurezza: capire da quali tipi di attacco bisogna difendersi (sezione 1.3.1);
- gestione della sicurezza: cercare di prevenire il verificarsi di questi attacchi (sezione 1.3.2).

1.3.1 Analisi della sicurezza



servizio di sicurezza un servizio di elaborazione o di comunicazione che è fornito da un sistema per dare uno specifico tipo di protezione alle risorse di sistema

asset l'insieme di beni, dati e persone necessarie all'erogazione di un servizio informatico

vulnerabilità debolezza di un asset

minaccia evento intenzionale o accidentale che può causare la perdita di una proprietà di sicurezza sfruttando una vulnerabilità

evento (negativo) verificarsi di una minaccia di tipo "evento accidentale"

attacco verificarsi di una minaccia di tipo "evento intenzionale"

incidente un evento di sicurezza che compromette l'integrità, la confidenzialità o la disponibilità di un asset informativo

breccia un incidente che comporta la diffusione o la potenziale esposizione di dati

diffusione di dati una breccia per la quale è stato confermato che i dati sono stati realmente diffusi (non solo esposti) a un party non autorizzato

hacker una persona che si diverte ad esplorare i dettagli dei sistemi programmabili e come allungarne le capacità, a differenza della maggioranza degli utenti, che preferiscono imparare solo lo stretto necessario

cracker una persona che sfrutta la propria conoscenza informatica per rompere la sicurezza in un sistema

Per un'analisi di sicurezza, innanzitutto è necessario definire il **servizio** che si vuole offrire agli utenti.

Tale servizio è erogato tramite degli **asset**:

- risorse ICT: CPU, cavi, reti, ecc.;
- dati: con questo termine si intendono i dati veri e propri, non i dischi su cui essi sono memorizzati: i dischi infatti potrebbero risultare del tutto integri, mentre i dati al loro interno potrebbero essere compromessi da accessi non autorizzati;
- location: è molto importante proteggere l'ubicazione fisica, perché i tipi di attacco che possono essere fatti dipendono anche da dove si trova la vittima;
- risorse umane: in un'azienda sono presenti alcune persone chiave che conoscono informazioni vitali per l'azienda (ad es. hanno la conoscenza per far funzionare specifici sistemi), e perciò devono essere ben tutelate, perché se per qualche motivo decidessero di lasciare l'azienda, nessuno saprebbe più far funzionare quei sistemi con conseguente perdita di denaro.

Una volta definiti gli asset, deve essere fatto un elenco di **eventi** possibili:

- vulnerabilità: ogni asset è caratterizzato da alcune vulnerabilità (ad es. location: i cavi di rete che attraversano una strada possono essere danneggiati da lavori di scavo);
- minacce: quali vulnerabilità possono essere sfruttate per effettuare un attacco?

Poiché la sicurezza di un sistema non può essere garantita al 100%, è necessario concentrarsi sulle minacce a priorità maggiore effettuando una **stima del rischio**:

- impatto: per quanto tempo il servizio smette di funzionare?
- probabilità dell'evento: quante volte si presenta una certa minaccia?

Origine degli attacchi

Per "costruire" la sicurezza, per prima cosa si deve identificare dove si trova il nemico:

- fuori dalla nostra organizzazione: è necessario costruire una difesa del perimetro (**firewall**);
- fuori dalla nostra organizzazione, con l'eccezione dei nostri partner: oltre al firewall, occorre creare una extranet sicura tramite una soluzione **VPN** per rendere sicure le comunicazioni con i partner;

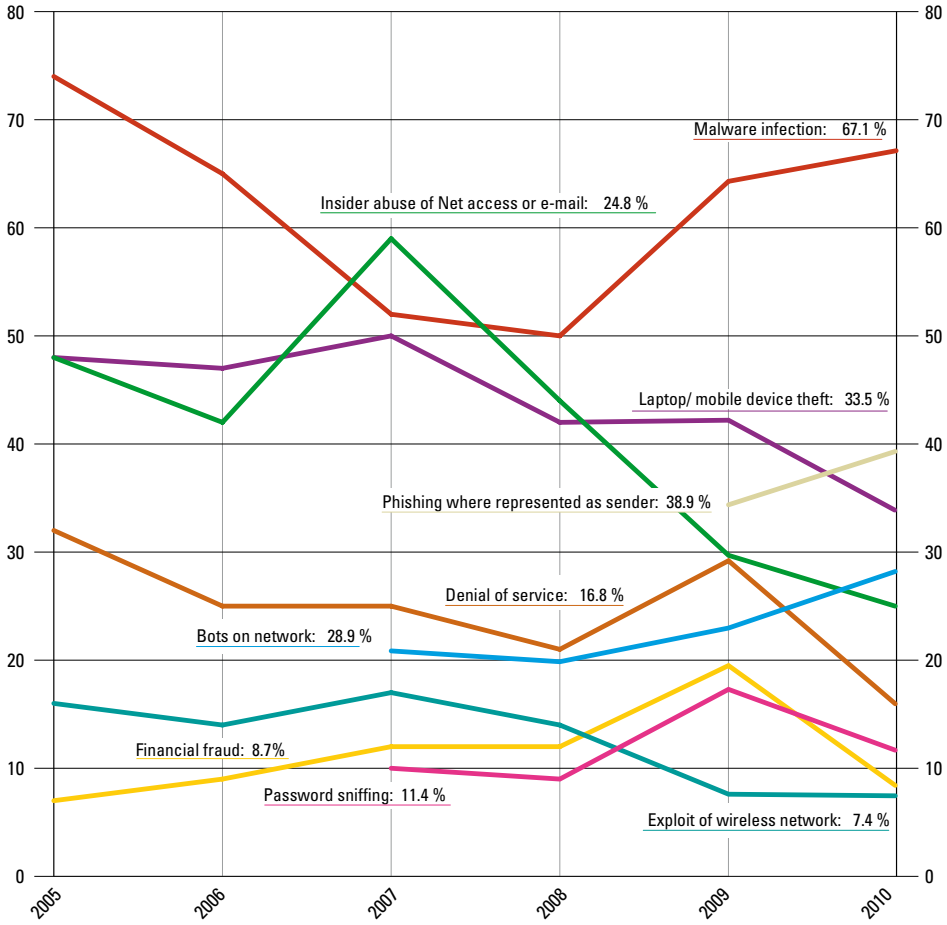


Figura 1.2: Evoluzione temporale dei principali attacchi rilevati dall'indagine annuale CSI/FBI.

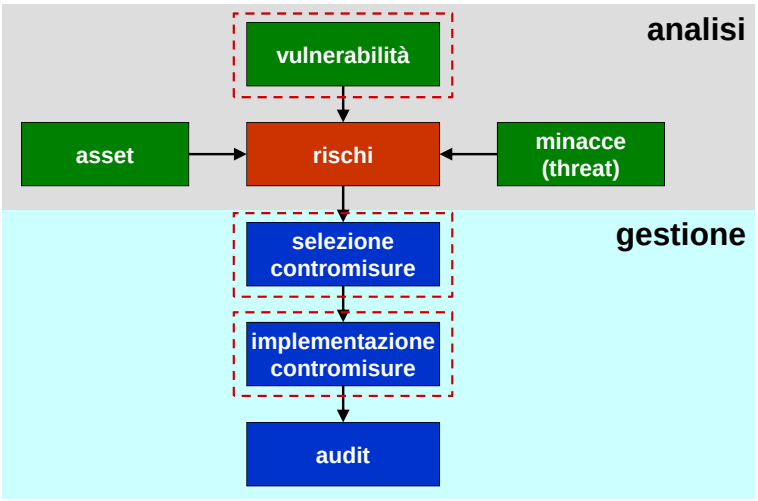
- dentro la nostra organizzazione: occorre implementare una protezione della intranet (LAN), ma questa è una contraddizione: una intranet viene creata per far collaborare le persone, mentre la sicurezza solitamente è un ostacolo alla collaborazione ⇒ è difficile trovare un adeguato compromesso tra il favorire la collaborazione e la necessità di protezione;
- ovunque: è l'ipotesi più realistica considerando la condizione attuale della rete, in cui gli utenti possono collegarsi da ovunque in qualsiasi momento ⇒ è meglio mettere la protezione **a livello applicativo**, che copre anche i dati, piuttosto che nella rete.

1.3.2 Gestione della sicurezza

Una volta che è stata effettuata una stima dei rischi del sistema, è necessario **gestire i rischi**:

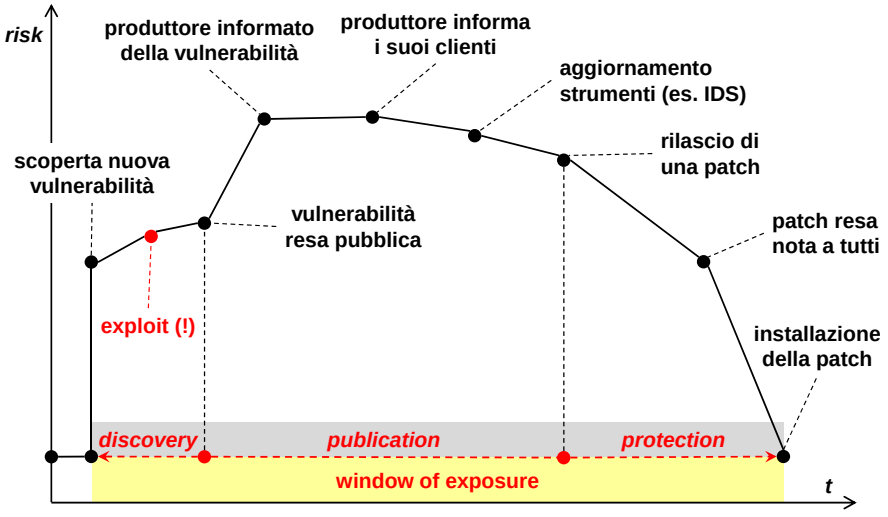
1. selezionare le contromisure da adottare per bloccare un certo rischio, a seconda dei costi e delle prestazioni;
2. implementare tali contromisure;
3. audit: verificare che le contromisure siano stato implementate correttamente e che funzionino come previsto (questo compito deve essere svolto periodicamente da una persona diversa dal progettista).

La sicurezza è un processo in corso, non un prodotto che si può comprare: le falle di sicurezza sono inevitabili, perciò il trucco è ridurre il rischio di esposizione a prescindere da prodotti o patch:



0. pianificazione: decidere la strategia di difesa (ad es. politiche di sicurezza);
1. prevenzione: implementare gli strumenti che cercano di evitare gli attacchi (ad es. firewall, VPN, PKI);
2. rilevamento: implementare gli strumenti che cercano di rilevare se un attaccante è riuscito a violare i sistemi di sicurezza (ad es. Intrusion Detection System [IDS], monitor di rete);
3. indagine: dopo ogni attacco, scoprire chi è l'attaccante e capire che cosa non ha funzionato per evitare che si ripeta (ad es. analisi forense, audit interno).
   Quest'ultima fase può portare a un cambiamento nella strategia della società includendo nuove minacce o modificando le contromisure esistenti.

1.3.3 Finestra di esposizione



La **finestra di esposizione** è il periodo di tempo in cui il sistema è esposto a una minaccia di sicurezza:

1. fase di scoperta: un attaccante sfrutta una vulnerabilità non nota:
   (a) un attaccante scopre una nuova vulnerabilità;

(b) l'attaccante sviluppa un programma di attacco, chiamato **exploit**, che sfrutta tale vulnerabilità;

(c) l'attaccante inizia ad effettuare degli attacchi, facendo sì che la vulnerabilità diventi pubblica;

2. fase di pubblicazione: le persone cercano di limitare i danni:
   (a) la vittima dell'attacco informa il produttore della vulnerabilità;
   (b) il produttore informa tutti i suoi clienti dell'esistenza di tale problema;
   (c) i clienti aggiornano gli strumenti di difesa (ad es. firme IDS), in attesa di una patch;
   (d) il produttore rilascia una **patch**;
3. fase di protezione: i clienti applicano la contromisura:
   (a) la patch è resa disponibile a tutti i clienti;
   (b) solo quando tutti installano la patch, il problema può essere considerato risolto.

1.4 Attacchi di base

- **source address spoofing**: gli indirizzi di rete sorgente nei pacchetti vengono falsificati (sezione 1.4.1);
- **packet sniffing**: password e/o dati riservati vengono letti da terzi non autorizzati (sezione 1.4.2);
- **DoS** e **DDoS**: il funzionamento di un servizio viene impedito o limitato (sezioni 1.4.3 e 1.4.4);
- **shadow server**: qualcuno si sostituisce a un host legittimo (sezione 1.4.5);
- **connection hijacking**: dati vengono inseriti o modificati durante il loro transito in rete (sezione 1.4.6).

1.4.1 Source address spoofing

masquerade un tipo di azione di minaccia per mezzo del quale un'entità non autorizzata ottiene l'accesso a un sistema o esegue un atto malevolo fingendosi illegittimamente un'entità autorizzata

spoof tentativo di un'entità non autorizzata di ottenere l'accesso a un sistema fingendosi un utente autorizzato

Il **source address spoofing** consiste nella falsificazione dell'indirizzo di rete (IP, MAC, ecc.) sorgente di un pacchetto.

Attacchi L'attaccante si finge qualcun altro (masquerade):

- falsificazione dei dati: si inviano i dati a nome di qualcun altro;
- accesso non autorizzato: si esegue l'accesso a un sistema utilizzando le credenziali di qualcun altro.

Contromisure Non usare mai autenticazione basata sugli indirizzi di rete.

1.4.2 Packet sniffing

wiretapping un attacco che intercetta le informazioni contenute in un flusso di dati e accede ad esse in un sistema di comunicazione

wiretapping passivo tenta solo di osservare il flusso di dati e ottenere la conoscenza delle informazioni contenute in esso

Il **packet sniffing** consiste nella lettura dei pacchetti destinati a un altro nodo della rete:

- reti broadcast (ad es. LAN): le comunicazioni avvengono tramite canali condivisi ⇒ si assume che chi non è il destinatario ignori i dati ricevuti;
- nodi di smistamento (ad es. switch, router): ogni apparato di rete ha tipicamente una porta di mirror per l'analisi del traffico.

Attacchi L'attaccante mette la sua scheda di rete in modalità promiscua per intercettare qualunque cosa (password, dati personali, ecc.).

Contromisure

- reti non broadcast: è praticamente impossibile realizzarle perché tutte le reti sono basate su questo principio;
- crittografia dei pacchetti: l'attaccante può leggere i dati, ma non li comprende; non si devono però criptare le informazioni (indirizzi e porte) contenute nell'intestazione usate dai nodi di smistamento.

1.4.3 DoS

denial of service l'impedimento dell'accesso autorizzato a una risorsa di sistema o il ritardamento delle operazioni e delle funzioni del sistema

L'attacco **denial-of-service** (DoS) consiste nel tenere un sistema impegnato a svolgere operazioni inutili in modo che non possa fornire i suoi servizi.

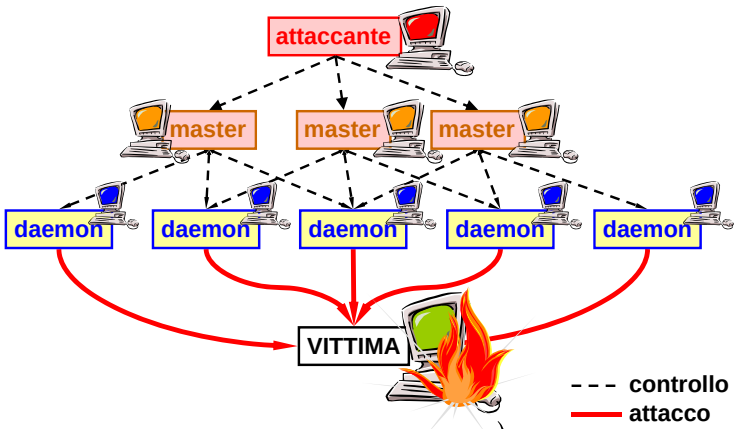
Attacchi L'attaccante impedisce l'uso di un sistema o di un servizio (disponibilità):

- saturazione della posta: si cerca di saturare la capacità della casella di posta elettronica in modo da rendere impossibile la ricezione di un messaggio importante;
- saturazione dei log: prima di effettuare un attacco, si manda una tale mole di richieste errate al server che la capacità di memorizzazione dei log sarà esaurita e le mosse dell'attaccante non saranno tracciate;
- ping flooding: si inviano in modo massivo messaggi ICMP Echo Request senza attenderne la risposta;
- SYN attack: si aprono delle connessioni TCP in sospeso senza mandare l'ACK di chiusura del three-way handshake, per saturare le tabelle dove si registrano le connessioni attive.

Contromisure Non esiste alcuna contromisura definitiva, ma solo dei rimedi palliativi basati sull'approccio quantitativo per mitigare gli effetti:

- monitoraggio: si cerca di rilevare eventuali anomalie (ad es. utilizzo delle risorse della CPU e di rete), anche se potrebbero essere originate da semplici malfunzionamenti;
- sovradimensionamento: si progetta il sistema per sopportare più carico di quello richiesto normalmente, così in caso di attacco DoS il sistema resiste per un po' di tempo, lasciando il tempo di identificare la sorgente dell'attacco.

1.4.4 DDoS



calcolo distribuito una tecnica che disperde un singolo insieme di compiti logicamente correlati tra un gruppo di computer geograficamente separati ma cooperanti

zombie un computer host Internet che è stato penetrato surrettiziamente da un intruso che ha installato software demone malevolo per far sì che l'host operi come un complice nell'attaccare altri host, specialmente negli attacchi distribuiti che tentano il denial of service tramite l'inondazione

L'attacco **distributed denial-of-service** (DDoS) moltiplica l'effetto dell'attacco DoS mettendo assieme più macchine:

1. l'attaccante individua la vittima;
2. l'attaccante sfrutta delle falle di sicurezza per installare del malware, con capacità di auto-aggiornamento, su molti nodi remoti, che diventano così degli **zombie** (o demoni o malbot), ignari di essere complici dell'attacco, e costituiscono una rete detta **botnet** (ad es. TrinOO, Tribe Flood Network [TFN], Stacheldraht [= filo spinato]);
3. l'attaccante elegge alcuni zombie a **master**, nodi con lo scopo di sincronizzare gli zombie nell'attacco (spesso tramite canali cifrati);
4. l'attaccante comunica ai master di lanciare l'attacco, quindi si scollega dalla rete in maniera che non sia possibile rintracciarlo, lasciando quindi che la botnet operi in autonomia;
5. i master comunicano agli zombie di lanciare l'attacco, tutti allo stesso istante, contro la vittima.

1.4.5 Shadow server

shadow server host che riesce a mostrarsi alle vittime come fornitore di un servizio senza averne il diritto

L'attacco **shadow server** consiste nel rispondere in maniera non autorizzata alle richieste a un servizio in vece del server legittimo. Richiede:

- packet sniffing: l'attaccante intercetta le richieste al servizio;
- address spoofing: l'attaccante risponde alle richieste come se fosse il server legittimo;
- DoS: il server ombra deve essere più veloce di quello reale a rispondere, in modo che le risposte del server reale arrivino dopo e siano scartate come pacchetti duplicati ⇒ il server legittimo può essere rallentato con un attacco DoS.

Attacchi La vittima non si accorge che sta comunicando con un altro server:

- fornitura di un servizio sbagliato: sono emesse delle risposte errate (ad es. un finto server DNS reindirizza a siti Web malevoli);
- furto di dati personali: la vittima fornisce i dati al servizio sbagliato.

Contromisure

- autenticazione del server: non bisogna fidarsi delle interfacce grafiche, degli indirizzi e dei nomi.

1.4.6 Connection hijacking

wiretapping attivo tenta di alterare i dati o di influenzare altrimenti il flusso

hijack attack una forma di wiretapping attivo in cui l'attaccante assume il controllo di un'associazione comunicativa precedentemente stabilita

attacco man-in-the-middle una forma di attacco di wiretapping attivo in cui l'attaccante intercetta e modifica selettivamente i dati comunicati per fingersi una o più delle entità coinvolte in un'associazione comunicativa

Il **connection hijacking** (= dirottamento) (o **data spoofing**) consiste nel prendere il controllo di un canale di comunicazione (già instaurato) e manipolare il traffico in transito inserendo, cancellando o modificando dei pacchetti:

- man-in-the-middle fisico: l'attaccante taglia il cavo e si mette fisicamente in mezzo;
- man-in-the-middle logico: l'attaccante fa sì che un nodo mandi i pacchetti a lui credendo di mandarli all'altro party, e poi egli li manda all'altro party (ad es. ARP poisoning: sezione 5.10.3).

Attacchi

- lettura, falsificazione e manipolazione dei dati scambiati tra due party.

Contromisure

- autenticazione: il connection hijacking prende il controllo del canale di comunicazione dopo che esso è stato instaurato ⇒ non è sufficiente l'autenticazione della controparte all'inizio della comunicazione, ma ogni singolo pacchetto di rete deve essere autenticato;
- integrità: permette di verificare se un pacchetto è stato modificato;
- serializzazione: non sono ammessi pacchetti fuori sequenza o mancanti (ad es. i comandi `cd /tmp` e `rm *` possono essere scambiati perdendo la loro innocuità); i numeri di sequenza non devono essere modificabili dagli attaccanti.

Man-in-the-browser

L'attacco man-in-the-middle (MITM) è sempre più difficile da compiere perché i canali di rete sono sempre più protetti ⇒ l'attacco **man-in-the-browser** (MITB) attacca direttamente i terminali degli utenti, che invece sono sempre meno protetti (ad es. smartphone), installando software (ad es. keylogger, estensione del browser) che manipola il traffico prima del canale idealmente sicuro.

1.5 Tecniche di social engineering

social engineering eufemismo per metodi non tecnici e a bassa tecnologia, spesso coinvolgenti trucchi o frode, che sono usati per attaccare i sistemi informativi

Le **tecniche di social engineering** chiedono, via posta elettronica, telefono o anche carta, la partecipazione inconsapevole dell'utente vittima all'azione d'attacco.

Problemi base (non tecnologici)

- scarsa comprensione del problema: spesso le persone non si rendono nemmeno conto che esiste un rischio di sicurezza;
- pressioni psicologiche: gli esseri umani sono soggetti alla fallibilità (soprattutto in condizioni di stress, sovraccarico, frustrazione, ecc.) e hanno una naturale tendenza alla fiducia;
- interfacce/architetture complesse: possono fuorviare l'utente e dare origine a comportamenti erronei;
- calo di prestazioni dovuto all'applicazione delle misure di sicurezza: spinge le persone a disabilitare le applicazioni di sicurezza diventando dei possibili bersagli.

1.5.1 Phishing

phishing una tecnica per tentare di acquisire dati sensibili, come numeri di conti bancari, attraverso una sollecitazione fraudolenta in un messaggio di posta elettronica o su un sito Web, in cui il perpetratore si finge una persona rispettabile o d'affari legittima

Il **phishing** consiste nell'attrarre (via posta elettronica, messaggio istantaneo, ecc.) l'utente di un servizio di rete su un server fasullo (shadow server) per:

- catturare credenziali di autenticazione o altre informazioni personali;
- convincerlo ad installare un plug-in o un'estensione che è in realtà un virus o un cavallo di Troia.

Il termine "phishing" deriva probabilmente da "phony fishing" (= pesca al gonzo): la sollecitazione coinvolge solitamente qualche tipo di esca per adescare i destinatari sprovveduti.

Esistono due varianti specializzate del phishing:

- **spear** (= lancia) **phishing**: si includono molti dati personali per aumentare la credibilità del messaggio (ad es. indirizzi di posta elettronica conosciuti, numeri di telefono reali, ecc.);
- **whaling** (= caccia alla balena): attacco mirato a colpire persone importanti (ad es. l'amministratore delegato) che sono a capo dell'azienda.

1.5.2 Pharming

pharming attacco mirato a reindirizzare il traffico di un sito Web a un altro sito Web fraudolento

Con **pharming** si intende l'insieme delle tecniche per reindirizzare un utente verso uno shadow server (ad es. cambiando gli indirizzi dei server DNS sul client):

- attacco diretto: si sfrutta una vulnerabilità o una malconfigurazione;
- attacco indiretto: si usano virus o worm.

1.6 Attacchi da malware

malware un qualsiasi software contenente codice malevolo

virus malware che, se eseguito dall'utente, provoca danni nel sistema; non è in grado di propagarsi ad altri sistemi, a meno che l'utente non trasferisce, anche involontariamente, il file infetto

worm malware che satura le risorse della CPU, della memoria o di rete del sistema creando delle repliche di se stesso, ed è in grado di propagarsi ad altri sistemi senza l'intervento dell'utente

cavallo di Troia il mezzo attraverso cui i malware vengono distribuiti solitamente

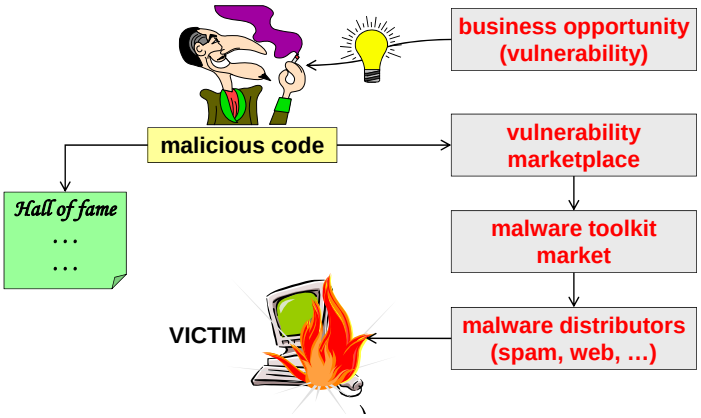
backdoor punto di accesso non autorizzato che può essere usato da un malware per aggirare il sistema di autenticazione ed entrare nel sistema

rootkit insieme di strumenti, nascosti nel sistema, per ottenere l'accesso al sistema con privilegi di amministratore (ad es. programma modificato, libreria, driver, modulo kernel, hypervisor)

I malware non sono solo un problema tecnologico, ma la loro propagazione richiede la complicità (anche involontaria) delle persone:

- utenti: sono quelli più facili da colpire, in quanto sono adescabili con parole chiave (gratis, urgente, non aprire questo...) ⇒ gli utenti vanno sensibilizzati sul problema della sicurezza e sull'importanza di installare un antivirus;
- sistemisti: possono configurare i sistemi in maniera errata per mancanza di tempo;
- produttori: possono inserire delle funzionalità che violano la sicurezza (ad es. esecuzione automatica, zone trusted).

1.6.1 Catena alimentare dei malware



In passato i cattivi sviluppavano malware per attaccare siti Web importanti (ad es. FBI) con il solo scopo di guadagnare fama, ma oggi essi preferiscono guadagnare soldi entrando nella catena alimentare dei malware:

1. scoperta di una vulnerabilità: il malware nasce sempre dalla scoperta di una nuova vulnerabilità in un sistema non ancora nota al pubblico (**vulnerabilità zero-day**);
2. vendita della vulnerabilità: le vulnerabilità scoperte vengono vendute all'asta al mercato nero ai creatori di malware toolkit, che hanno bisogno di sapere più vulnerabilità possibili per essere avanti rispetto ai produttori di antivirus;

3. vendita del malware toolkit: i malware toolkit che sfruttano le vulnerabilità vengono venduti ai distributori di malware, che gestiscono le reti di spam o i siti Web per distribuire malware;
4. vendita della rete di distribuzione del malware: l'attaccante finale acquista la rete da un distributore e la usa per attaccare la vittima.

1.6.2 Zeus

Zeus è un malware scoperto nel 2007 (non si sa quando sia nato) e venduto (forse) al miglior offerente nel 2010. **Zbot** è il nome della botnet di zombie su cui Zeus è installato (stima: 3,6 milioni di copie attive solo negli USA).

Zeus può essere usato:

- direttamente da chi ne ha il controllo:
  - MITB per keylogging: si registrano tutti i tasti premuti dall'utente;
  - form grabbing: si acquisiscono i dati inseriti all'interno di un form;
- indirettamente come un mediatore, per caricare altro malware:
  - CryptoLocker ransomware: il disco fisso della vittima viene criptato, e per avere la chiave per decriptarlo la vittima deve pagare del denaro.

Zeus è molto difficile da scoprire e da rimuovere, perché usa una serie di tecniche stealth per nascondersi nel sistema.

1.6.3 Stuxnet

Stuxnet è un malware che è una combinazione di worm + virus per sistemi Windows. Costituisce il prototipo di un nuovo tipo di attacco, perché è stato il primo caso noto di attacco verso i sistemi SCADA, che sono sistemi usati per controllare gli impianti industriali.

Mimetizzandosi come driver di Windows (apparentemente firmato da Microsoft), nel 2010 ha attaccato il sistema informatico che controllava le centrali nucleari in Iran sfruttando 4 vulnerabilità presenti in servizi di Windows che erano abilitati senza necessità:

- 2 vulnerabilità zero-day;
- 1 vulnerabilità nota e con patch disponibile (ma non installata);
- 1 vulnerabilità nota ma senza patch disponibile.

Siccome i computer non erano collegati a Internet, il primo veicolo di infezione è stata probabilmente una chiavetta USB dei tecnici di manutenzione, e poi il malware si è propagato tramite i dischi di rete condivisi senza alcuna protezione.

L'attacco ha avuto successo a causa di aspetti che fino a quel momento non erano considerati dei rischi di sicurezza, dal momento che il sistema era protetto fisicamente da guardie armate e non era collegato a Internet:

- mancanza delle protezioni standard (antivirus, firewall...): l'ambiente era considerato trusted;
- mancanza degli aggiornamenti periodici di Windows: il servizio Windows Update ha bisogno di scaricare gli aggiornamenti da Internet;
- malconfigurazione: erano attivi alcuni servizi non necessari (MS-RPC, MS-spool, dischi di rete condivisi);
- autenticazione trascurata: la password di accesso al database di back-end era quella predefinita, identica per tutti i sistemi;
- mancanza di una lista di validazione: qualsiasi software poteva essere installato su quei computer.

Fratelli di Stuxnet

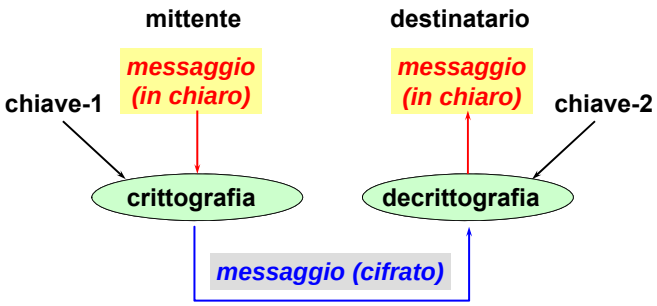
Alcuni malware fratelli di Stuxnet che condividono la stessa piattaforma di sviluppo (tilded) sono:

- **Duqu** (2011): non è né un worm né un virus, ma svolge attività di ricognizione e di intelligence: quando installato, invia all'esterno tutte le informazioni di sistema utili per guidare un attacco;
- **Flame** (2012): è in grado di spiare i sistemi registrando traffico di rete, audio, video e tastiera, si diffonde tramite chiavette USB o tramite rete, senza mai causare danni fisici, e contiene una backdoor per consentire la configurazione e l'aggiornamento da remoto.

Capitolo 2

Concetti base di sicurezza informatica

2.1 Concetti base della crittografia



crittografia la disciplina che incorpora i principi, i mezzi e i metodi per la trasformazione dei dati al fine di nascondere il loro contenuto informativo [...]

criptazione la trasformazione crittografica dei dati per produrre il testo cifrato

cifrario un algoritmo crittografico per la criptazione e la decriptazione

testo in chiaro dati intelligibili, il cui contenuto semantico è disponibile

testo cifrato dati prodotti attraverso l'uso della cifratura; il contenuto semantico dei dati risultanti non è disponibile

chiave un parametro in ingresso usato per variare una funzione di trasformazione effettuata da un algoritmo crittografico

La **criptazione** utilizza un algoritmo matematico che trasforma dei dati in chiaro P in un testo cifrato C tramite una chiave di cifratura K_1, al fine di impedire che quei dati possano essere compresi da chi non è autorizzato:

$$C = \mathrm{enc}\,(K, P) = \{P\}\,K_1$$

La **decriptazione** permette di svolgere il processo inverso tramite una chiave di decifratura K_2:

$$P = \mathrm{dec}\,(K_2, C) = \mathrm{enc}^{-1}\,(K_2, C)$$

Le chiavi K_1 e K_2 sono legate da una relazione matematica e devono sempre essere usate in coppia: chi non possiede la chiave di decifratura K_2 può leggere i dati cifrati, ma non può comprenderli in quanto una chiave di decifratura sbagliata porta a un output errato.

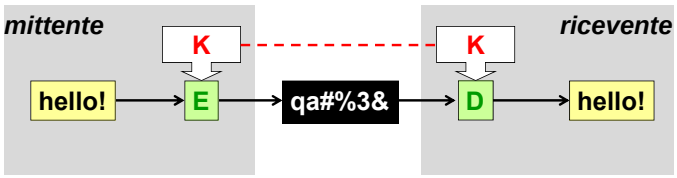
Valutazione delle prestazioni In generale le prestazioni crittografiche non dipendono dalla quantità di RAM, ma essenzialmente dalla CPU: architettura, instruction set (istruzioni già disponibili in hardware), dimensioni della cache.

Le prestazioni comunque non sono tipicamente un problema sui client, a meno che non siano sovraccaricati dalle applicazioni locali o non siano sistemi embedded con risorse di calcolo limitate, ma possono essere un problema sui server e sui nodi di rete (ad es. router) che devono svolgere le operazioni di crittografia per molti client.

Gli **acceleratori crittografici** sono delle schede che implementano in hardware alcune operazioni crittografiche per migliorare le prestazioni:

- generici: implementano solo un algoritmo di crittografia (ad es. SHA-3);
- specifici: implementano un intero protocollo di sicurezza (ad es. SSL, IPsec).

2.1.1 Crittografia simmetrica



crittografia simmetrica una branca della crittografia in cui gli algoritmi usano la stessa chiave per entrambe le due operazioni crittografiche delle controparti (ad es. criptazione e decriptazione)

chiave simmetrica una chiave crittografica che è usata in un algoritmo crittografico simmetrico

Nella **crittografia simmetrica** (o **crittografia a chiave segreta**), la chiave simmetrica è una chiave unica K, sia per la criptazione sia per la decriptazione, condivisa solo tra il mittente e il ricevente:

$$K_1 = K_2 = K \Rightarrow \begin{cases} C = \text{enc}\,(K, P) = \{P\}\,K \\ P = \text{dec}\,(K, C) = \text{enc}^{-1}\,(K, C) \end{cases}$$

La crittografia simmetrica richiede un carico di elaborazione minore rispetto alla crittografia asimmetrica $\Rightarrow$ è particolarmente adatta per criptare grosse quantità di dati.

Gli algoritmi simmetrici si suddividono in due classi:

- **algoritmi a blocchi** (ad es. DES, IDEA, RC2, RC5, AES): suddividono i dati da criptare in blocchi di eguale dimensione, quindi elaborano un blocco alla volta (sezione 2.4);
- **algoritmi di tipo stream** (ad es. RC4, SEAL): sono in grado di operare su un flusso di dati, un bit o un byte alla volta (sezione 2.5).

2.1.2 Crittografia asimmetrica

crittografia asimmetrica una moderna branca della crittografia (nota popolarmente come "crittografia a chiave pubblica") in cui gli algoritmi usano una coppia di chiavi (una chiave pubblica e una chiave privata) e usano un componente diverso della coppia per ognuna delle due operazioni crittografiche delle controparti (ad es. criptazione e decriptazione, o creazione della firma e verifica della firma)

chiave asimmetrica una chiave crittografica che è usata in un algoritmo crittografico asimmetrico

coppia di chiavi un insieme di chiavi correlate matematicamente – una chiave pubblica e una

chiave privata – che sono usate per la crittografia asimmetrica e sono generate in un modo che rende computazionalmente infattibile derivare la chiave privata dalla conoscenza della chiave pubblica

chiave pubblica la componente pubblicamente divulgabile di una coppia di chiavi crittografiche usata per la crittografia asimmetrica

chiave privata la componente segreta di una coppia di chiavi crittografiche usata per la crittografia asimmetrica

Nella **crittografia asimmetrica** (o **crittografia a chiave pubblica**), la chiave asimmetrica è costituita da una coppia di chiavi K_{pri} e K_{pub}, distinte in base al modo in cui vengono conservate:

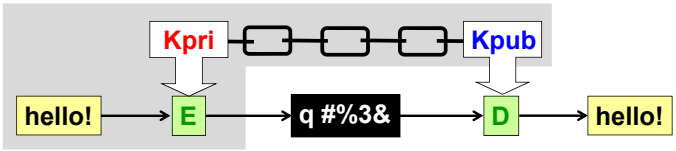
- la **chiave privata** K_{pri} è conosciuta solo da chi la possiede, ed è tenuta segreta al resto del mondo;
- la **chiave pubblica** K_{pub} è diffusa il più ampiamente possibile al resto del mondo.

Caratteristiche delle chiavi

- ruoli interscambiabili: le chiavi non sono distinte in base al ruolo, ma solo in base al modo in cui vengono conservate, poiché ciascuna di esse può essere usata sia per criptare sia per decriptare;
- funzionalità reciproca: dati criptati con una chiave possono essere decriptati solo con l'altra nella stessa coppia.

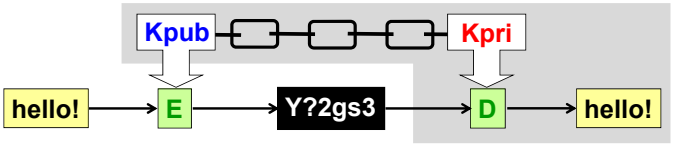
Nella trasmissione di dati da un mittente a un ricevente, possono essere ottenute due importanti proprietà di sicurezza:

- **autenticazione dell'origine dati**: viene usata la chiave asimmetrica del mittente:



  1. il mittente cripta il testo in chiaro con la sua chiave privata K_{pri};
  2. il ricevente decripta il testo cifrato con la chiave pubblica K_{pub} del mittente: l'output della decriptazione con la chiave pubblica è corretto solo se per la criptazione è stata usata la chiave privata corrispondente $\Rightarrow$ il ricevente ha la prova che i dati sono stati criptati veramente dal mittente, l'unico che ha potuto usare la chiave privata;

- **riservatezza senza segreti condivisi**: viene usata la chiave asimmetrica del ricevente:



  1. il mittente cripta il testo in chiaro con la chiave pubblica K_{pub} del ricevente;
  2. il ricevente decripta il testo cifrato con la sua chiave privata K_{pri}: l'output della decriptazione con la chiave privata è corretto solo se per la criptazione è stata usata la chiave pubblica corrispondente $\Rightarrow$ il mittente ha la garanzia che i dati saranno decriptati solo dal ricevente, l'unico che potrà usare la chiave privata.

La crittografia asimmetrica è decisamente più complessa rispetto alla crittografia simmetrica: è stata resa possibile solamente alla fine del secolo scorso grazie a notevoli progressi nel campo della matematica. Anche per gli attuali sistema di elaborazione, la crittografia asimmetrica presenta notevoli difficoltà di calcolo, risultando molto più lenta rispetto alla crittografia simmetrica. Per questo motivo, la crittografia asimmetrica è adatta per criptare piccole quantità di dati (qualche decina di bit), principalmente per:

- firma digitale: sfrutta l'autenticazione dell'origine dati (sezione 2.11.1);
- distribuzione in-band delle chiavi segrete: sfrutta la riservatezza senza segreti condivisi (sezione 2.3.1).

2.2 Resistenza di un algoritmo di crittografia

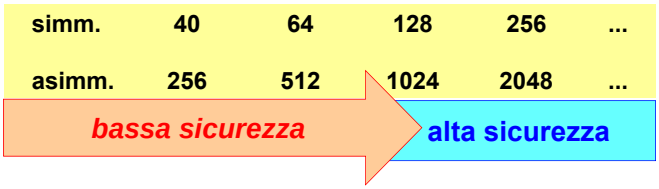


Figura 2.1: La continua evoluzione nel tempo della potenza di calcolo richiede chiavi sempre più lunghe.

resistente usato per descrivere un algoritmo crittografico che richiederebbe una grande quantità di potenza computazionale per sconfiggerlo

lunghezza della chiave il numero di simboli (di solito dichiarato come un numero di bit) necessario per poter rappresentare uno qualsiasi dei valori possibili di una chiave crittografica

lunghezza della chiave efficace una misura della resistenza di un algoritmo crittografico, indipendentemente dalla lunghezza effettiva della chiave

security by obscurity tentare di mantenere o aumentare la sicurezza di un sistema tenendo segrete la progettazione o la costruzione di un meccanismo di sicurezza

Un algoritmo di crittografia è tanto più **resistente** quanto più è difficile per un eventuale attaccante decriptare del testo cifrato.

La lunghezza delle chiavi deve essere un compromesso, in base alla potenza di calcolo disponibile sui sistemi correnti e al livello di segretezza del messaggio criptato, tra:

- complessità: la criptazione e la decriptazione non devono durare troppo tempo;
- sicurezza: un attacco a forza bruta non deve durare troppo poco tempo.

Principio di Kerckhoffs La resistenza di un algoritmo si ottiene non tenendolo segreto, ma divulgandolo affinché sia analizzato il più possibile dalla comunità di ricerca crittografica per migliorare la sua progettazione. Se le chiavi:

- sono tenute segrete: l'utente deve ricordarle a memoria o conservarle in un luogo sicuro;
- sono gestite solo da sistemi fidati: un malware installato su di essi può rubarle facilmente;
- sono di lunghezza adeguata: l'attacco a forza bruta non deve essere fattibile, cioè non deve durare troppo poco tempo;

allora non ha alcuna importanza che l'algoritmo di crittografia sia tenuto segreto, anzi è bene che sia pubblico affinché sia studiato ampiamente e siano rivelate eventuali debolezze.

2.2.1 Resistenza degli algoritmi simmetrici

Singola criptazione

Se:

- la chiave efficace, di lunghezza N bit, soddisfa le ipotesi del principio di Kerckhoffs;
- l'algoritmo di crittografia è stato ben progettato;

allora l'unico attacco possibile è l'**attacco a forza bruta** (o attacco esaustivo), che richiede un numero T di tentativi esponenziale nella lunghezza N della chiave:

$$T = 2^N$$

Numero pari di criptazioni

Un algoritmo di criptazione non si applica mai un numero pari di volte. Infatti, l'idea di applicare due volte l'operazione di criptazione, anche con due chiavi K_1 e K_2 differenti:

$$C = \text{enc}\left(K_2, \text{enc}\left(K_1, P\right)\right)$$

è un grosso sbaglio: l'applicazione doppia di algoritmi di criptazione è soggetta ad un attacco di tipo known-plaintext, chiamato **meet-in-the-middle**, che consente di trovare le chiavi K_1 e K_2 applicando due volte la tecnica a forza bruta $\Rightarrow$ nonostante il tempo di calcolo raddoppi, la lunghezza della chiave efficace aumenta di un solo bit, guadagnando un solo "bit di sicurezza":

$$T = 2 \cdot 2^N = 2^{N+1}$$

Supponendo di conoscere:

- la lunghezza N delle due chiavi K_1 e K_2;
- il testo in chiaro P;
- il testo cifrato C;

l'attaccante può compiere l'attacco meet-in-the-middle in questo modo:

1. cripta il testo in chiaro P con tutte le possibili chiavi di cifratura K_i;
2. decripta il testo cifrato C con tutte le possibili chiavi di decifratura K_j;
3. confronta gli output per trovare l'output tale che:

$$\text{dec}\left(K_2, C\right) = \text{enc}\left(K_1, P\right)$$

4. scarta facilmente le coppie (K_i, K_j) false positive se conosce più di una coppia (P, C).

Questo è il motivo per cui non esiste il doppio DES.

Numero dispari di criptazioni

Un algoritmo di criptazione si può applicare un numero dispari di volte solo se l'algoritmo simmetrico di base non è un gruppo (dal punto di vista matematico). Se l'algoritmo è un gruppo, allora la sua resistenza non cambia, perché esiste sempre una chiave K tale che:

$$\text{enc}\left(K_3, \text{enc}\left(K_2, \text{enc}\left(K_1, P\right)\right)\right) = \text{enc}\left(K, P\right)$$

Siccome il DES non è un gruppo, il 3DES può essere applicato in modalità Encrypt-Encrypt-Encrypt (EEE) aumentando la sua resistenza (si rimanda alla sezione 2.6.2).

2.2.2 Resistenza degli algoritmi asimmetrici

Lunghezza delle chiavi pubbliche

Le chiavi asimmetriche devono essere un ordine di grandezza più lunghe di quelle simmetriche perché:

- la chiave pubblica porta con sé alcune informazioni sulla chiave privata;
- non tutte le chiavi sono valide, ma le chiavi seguono delle regole matematiche ben definite.

La lunghezza delle chiavi ottimale è stata testata tramite delle sfide RSA, il cui obiettivo era quello di riuscire a fattorizzare il risultato nel minor tempo possibile. È stato dimostrato che:

- 512 bit sono attaccabili in alcune settimane;
- 1024 bit sono attaccabili in alcuni mesi;
- 2048 bit offrono un livello di sicurezza ragionevole per vari anni.

Complessità

Gli algoritmi di crittografia asimmetrica sono basati su problemi matematicamente difficili da risolvere per i cattivi, ma facili da applicare per i buoni:

- RSA: il buono deve calcolare il prodotto di due numeri primi, mentre il cattivo deve calcolare la fattorizzazione del risultato (cioè ricavare i numeri primi):

$$N = P \cdot Q \Rightarrow \begin{cases} P = {}^{N}/_{Q} \\ Q = {}^{N}/_{P} \end{cases}$$

- DH, DSA: il buono deve calcolare l'esponenziazione discreta, mentre il cattivo deve calcolare il logaritmo discreto del risultato (cioè ricavare la base e l'esponente):

$$\begin{cases} A = g^x \bmod p \\ B = g^y \bmod p \end{cases} \Rightarrow \begin{cases} x = \log_g A \\ y = \log_g B \end{cases}$$

Non sono attualmente noti algoritmi polinomiali in grado di risolvere i problemi matematici dei cattivi in tempo ragionevole con i moderni calcolatori:

- fattorizzazione: tutti gli algoritmi noti non sono polinomiali;
- logaritmo discreto (spesso adattabili anche per la fattorizzazione):
  - moltiplicazioni successive: impiegano un tempo lineare in p, e quindi esponenziale nel suo numero di bit;
  - algoritmo di Shor: è polinomiale ($\sim O\left(\log^3 N\right)$), è in grado di risolvere RSA e DH, ma richiede un computer quantistico;
  - altri algoritmi (ad es. Pohlig-Hellman, number field sleve): sono migliori, ma non sono polinomiali.

2.3 Distribuzione delle chiavi crittografiche

2.3.1 Distribuzione della chiave segreta

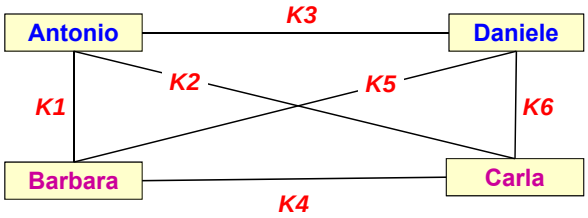


Figura 2.2: Le comunicazioni tra 4 party richiedono la distribuzione di 6 chiavi.

key establishment una procedura che combina le fasi di generazione della chiave e di distribuzione della chiave necessarie per instaurare o installare un'associazione comunicativa sicura

generazione della chiave un processo che crea la sequenza di simboli che compongono una chiave crittografica

distribuzione della chiave un processo che consegna una chiave crittografica dalla posizione dove viene generata alle posizioni dove viene usata in un algoritmo crittografico

out-of-band trasferimento di informazioni utilizzando un canale o un metodo che è fuori da (ad es. separato da o diverso da) il canale principale o il metodo normale

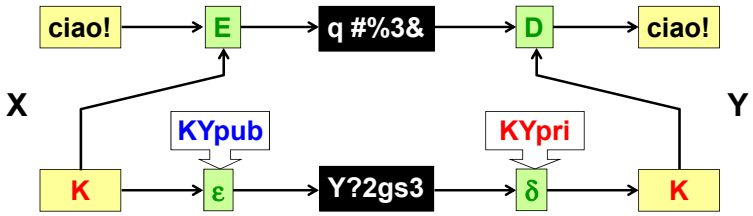
Dati N party che vogliono comunicare in modo sicuro a coppie, il numero n di chiavi necessarie per una comunicazione privata completa cresce linearmente con il numero N di party al quadrato:

$$n = \frac{N\,(N-1)}{2} = O\left(N^2\right)$$

Una volta scelta una chiave segreta, esistono due strategie per consegnarla all'altro party in modo sicuro:

- distribuzione out-of-band (OOB): la chiave è trasmessa su un canale diverso da quello usato normalmente per i messaggi criptati (ad es. canale normale = Internet, canale alternativo = telefono);
- distribuzione in-band: la chiave è scambiata per mezzo di un algoritmo di crittografia asimmetrico (pagina 35);
- key agreement: la chiave è concordata per mezzo di un algoritmo di key agreement (pagina 36).

Scambio della chiave segreta mediante algoritmi asimmetrici



trasporto della chiave un metodo di key establishment tramite il quale una chiave segreta viene generata da un'entità di sistema in un'associazione comunicativa e inviata in modo sicuro a un'altra entità nell'associazione

La riservatezza senza segreti condivisi viene spesso usata per comunicare in modo sicuro la chiave segreta scelta per un algoritmo simmetrico:

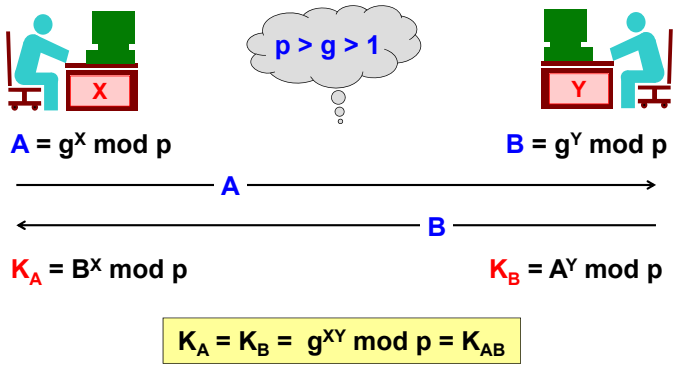
1. il mittente X sceglie una chiave K;
2. X cripta i dati usando la chiave K tramite l'algoritmo simmetrico E;
3. X cripta la chiave K usando la chiave pubblica $K_{Y\,\text{pub}}$ del ricevente Y;
4. X invia la chiave criptata e i dati criptati a Y;
5. Y decripta la chiave K usando la sua chiave privata $K_{Y\,\text{pri}}$;
6. Y decripta i dati usando la chiave K tramite l'algoritmo simmetrico D.

In questo schema ci sono due catene di trasmissione e di criptazione:

- catena simmetrica: usata per criptare velocemente e trasmettere una grossa quantità di dati;
- catena asimmetrica: usata per criptare e trasmettere una piccola quantità di dati, cioè la chiave simmetrica.

Nella pratica, questo metodo di trasmissione dei dati potrebbe non essere ritenuto sicuro, soprattutto in ambito militare: la chiave simmetrica viene scelta arbitrariamente dal mittente $\Rightarrow$ il ricevente deve fidarsi che il mittente non abbia rivelato la chiave a terzi (ad es. nemici).

Diffie-Hellman



key agreement un metodo per negoziare un valore di chiave in linea senza trasferire la chiave, neanche in forma criptata, ad es. la tecnica Diffie-Hellman

Diffie-Hellman (DH) è stato il primo algoritmo ad usare il concetto di pubblicare qualche cosa (i numeri p e g), e quindi in questo senso spesso è definito come il primo algoritmo a chiave pubblica inventato.

DH non è un metodo per distribuire la chiave, ma per concordare la chiave:

1. il mittente X e il ricevente X scelgono o concordano due numeri interi pubblici, un generatore g (tipicamente 2, 3 o 5) e un numero primo p (grande), tali che:

$$p > g > 1$$

2. X sceglie arbitrariamente un numero intero $x > 0$ (grande), e calcola:

$$A = g^x \bmod p$$

3. Y sceglie arbitrariamente un numero intero $y > 0$ (grande), e calcola:

$$B = g^y \bmod p$$

4. X invia (pubblica) il valore calcolato A a Y;
5. Y invia (pubblica) il valore calcolato B a X;
6. X riceve B, e calcola:

$$K_A = B^x \bmod p$$

7. Y riceve A, e calcola:

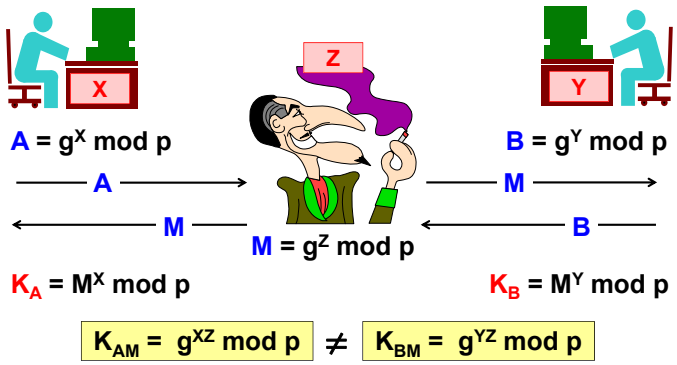
$$K_B = A^y \bmod p$$

8. per la proprietà delle potenze, i valori K_A e K_B, calcolati indipendentemente da X e Y, sono uguali e costituiscono la chiave segreta K_{AB}:

$$\begin{cases} K_A = (g^y)^x \bmod p \\ K_B = (g^x)^y \bmod p \end{cases} \Rightarrow K_A = K_B = g^{xy} \bmod p = K_{AB}$$

Entrambe le parti sono coinvolte direttamente nella scelta della chiave ⇒ questo metodo può essere usato quando le parti non si fidano l'uno dell'altro e non accettano che uno dei due inventi la chiave: il mittente non può aver rivelato precedentemente a terzi la chiave segreta perché essa è creata al momento stesso della comunicazione.

DH è resistente agli attacchi di sniffing: un man in the middle può intercettare A e B in transito, che sono calcolati esclusivamente sulla base di dati pubblici, ma non conosce i dati segreti x e y e quindi non può calcolare la chiave segreta K_{AB}.



Attacco man-in-the-middle attivo Nel caso in cui però l'attaccante Z sia in grado di manipolare i dati in transito, può allora sniffare il traffico tra le parti manipolando il processo di key agreement:

1. X e Y scelgono x e y, e calcolano A e B;
2. Z sceglie z e, conoscendo p e q perché pubblici, calcola M:

$$\begin{cases} A = g^x \bmod p \\ B = g^y \bmod p \\ M = g^z \bmod p \end{cases}$$

3. X invia il valore calcolato A a Y;
4. Z sostituisce A con M, e lo invia a Y;
5. Y invia il valore calcolato B a X;
6. Z sostituisce B con M, e lo invia a X;
7. X riceve M, e calcola:
$$K_A = M^x \bmod p$$
8. X riceve M, e calcola:
$$K_B = M^y \bmod p$$
9. i valori K_A e K_B non sono più uguali, e per di più sono noti all'attaccante Z:
$$\begin{cases} K_A = (g^z)^x \bmod p \\ K_B = (g^z)^y \bmod p \end{cases} \Rightarrow K_A = g^{xz} \bmod p \neq K_B = g^{yz} \bmod p$$

L'attaccante Z è riuscito in questo modo a svolgere due key agreement separati:

- un key agreement con X, concordando la chiave K_A;
- un key agreement con Y, concordando la chiave K_B.

Ogni volta che X manda dei dati a Y, questi possono essere sniffati da Z senza che le due parti se ne accorgano:

1. X cripta i dati con la chiave K_A e la manda a Y;
2. Z intercetta i dati criptati e li decripta con la chiave K_A;
3. Z può leggere e anche modificare i dati come desidera;
4. Z cripta i dati con la chiave K_B e li manda a Y;
5. Y decripta i dati ricevuti con la chiave K_B.

Questa operazione risulta del tutto trasparente al mittente e al ricevente: essi non hanno modo di sapere che al di là del canale protetto non c'è la controparte ma c'è l'attaccante, le chiavi che credono di aver concordato sono diverse, e i dati sono stati manipolati. È quindi necessario proteggere le chiavi scambiate tramite un processo di pre-autenticazione usando:

- certificati per le chiavi DH;
- protocolli autenticati varianti del DH (ad es. MQV: inventato da Menezes-Qu-Vanstone e brevettato da CertiCom).

2.3.2 Distribuzione della chiave pubblica

La distribuzione della chiave pubblica richiede alcune garanzie sulla corrispondenza tra la chiave pubblica e l'identità della persona:

- autenticità: la chiave pubblica appartiene effettivamente a chi dichiara esserne il proprietario?
- integrità: la chiave pubblica ha subito modifiche dopo che il suo proprietario l'ha pubblicata?

Le chiavi pubbliche possono quindi essere distribuite in due modi sicuri:

- scambio di chiavi out-of-band: la chiave è scambiata su un canale alternativo, analogamente alla distribuzione della chiave segreta;
- distribuzione per mezzo di un certificato a chiave pubblica: è un certificato di identità digitale (si rimanda alla sezione 3.1).

2.4 Algoritmi simmetrici a blocchi

cifrario a blocchi un algoritmo di criptazione che rompe il testo in chiaro in segmenti di dimensione fissa e usa la stessa chiave per trasformare ciascun segmento in chiaro in un segmento di testo cifrato di dimensione fissa

Due concetti sono alla base della progettazione dei moderni algoritmi simmetrici a blocchi:

- confusione: garantisce che una modifica al testo in chiaro porti a un'alterazione nel testo cifrato che è impossibile da predire;
- diffusione: garantisce che una modifica al testo in chiaro porti a un'alterazione nel testo cifrato che è diffusa.

Operatore XOR L'operatore di confusione ideale è l'operatore XOR, l'operatore binario meglio adatto alle specifiche di crittografia: è in grado di produrre un output con la stessa distribuzione di probabilità dei valori 0 e 1 dell'input, cioè dato un input binario casuale (ogni bit può valere 0 con probabilità 50% o 1 con probabilità 50%) l'output sarà casuale allo stesso modo $\Rightarrow$ l'output non suggerisce a un eventuale attaccante alcuna informazione utile circa la distribuzione dell'input. Inoltre lo XOR è un'operazione primitiva molto rapida dal punto di vista dell'hardware, ed è incorporata in tutte le CPU.

Modalità di applicazione Un algoritmo a blocchi come si può applicare a dati che hanno una dimensione diversa da quella del blocco base?

modalità operativa una tecnica per migliorare l'effetto di un algoritmo crittografico o per adattare l'algoritmo per un'applicazione, come applicare un cifrario a blocchi a una sequenza di blocchi di dati o a un flusso di dati

- dati > blocco base: ECB (sez. 2.4.1), CBC (sez. 2.4.2), IGE (sez. 2.12.2), combinati con padding (sez. 2.4.3) o CTS (sez. 2.4.4) quando i dati non sono multipli del blocco base;
- dati < blocco base (ad es. trasmissioni su linee seriali o parallele): CFB (sez. 2.4.5), OFB (sez. 2.4.6), CTR (sez. 2.4.7).

È estremamente necessario scegliere correttamente la modalità di applicazione di un algoritmo, altrimenti si è attaccabili a prescindere dalla resistenza dell'algoritmo utilizzato.

2.4.1 ECB

(a) Criptazione. *(b) Decriptazione.*

La modalità **Electronic Code Book** (ECB) consiste in:

1. suddividere il testo in chiaro da criptare in blocchi base di dimensione fissa;
2. applicare l'algoritmo di criptazione su ciascun blocco i-esimo, indipendentemente dagli altri blocchi:
$$C_i = \text{enc}\,(K, P_i)$$
3. concatenare i blocchi risultanti per produrre il testo cifrato.

La decriptazione avviene invertendo la sequenza di operazioni:

$$P_i = \text{enc}^{-1}\,(K, C_i)$$

Vantaggi

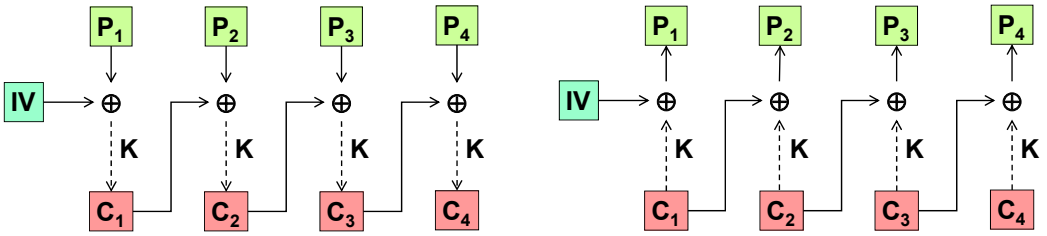
- no propagazione degli errori: un errore nella trasmissione provoca un errore nella decriptazione di un solo blocco senza propagarsi agli altri blocchi;
- parallelizzazione della CPU: è possibile lavorare su più blocchi allo stesso tempo.

Svantaggi Quando la quantità di dati è maggiore della dimensione del blocco base (c'è più di 1 blocco), la criptazione di ogni blocco è indipendente dagli altri blocchi:

- attacchi di swapping: la criptazione non dipende dalla posizione del blocco ⇒ è possibile scambiare due blocchi, o spostare un blocco, qualunque senza che ciò venga scoperto;
- attacchi known-plaintext: blocchi identici sono criptati allo stesso modo ⇒ un attaccante può costruire un database contenente gli output delle criptazioni con tutte le possibili chiavi di un blocco ricorrente (ad es. intestazione di un documento di Word), per poi trovare la chiave confrontando ogni blocco del testo cifrato alla ricerca del blocco ricorrente (pattern).

attacco known-plaintext una tecnica di crittoanalisi in cui l'analista prova a determinare la chiave dalla conoscenza di alcune coppie testo in chiaro-testo cifrato (anche se l'analista potrebbe anche avere altri indizi, come la conoscenza dell'algoritmo crittografico)

2.4.2 CBC



(a) Criptazione. *(b) Decriptazione.*

valore di inizializzazione (IV) un parametro in ingresso che imposta lo stato di partenza di una modalità o algoritmo crittografico

La modalità **Cipher Block Chaining** (CBC) consiste in:

1. suddividere il testo in chiaro da criptare in blocchi base di dimensione fissa;
2. applicare l'algoritmo di criptazione su ciascun blocco i-esimo, mettendo in XOR il risultato della criptazione del blocco precedente $i-1$-esimo:
$$C_i = \text{enc}\left(K, P_i \oplus C_{i-1}\right)$$
3. concatenare i blocchi risultanti per produrre il testo cifrato.

La decriptazione avviene invertendo la sequenza di operazioni (lo XOR è l'inverso di se stesso):

$$P_i = \text{enc}^{-1}\left(K, C_i\right) \oplus C_{i-1}$$

Vettore di inizializzazione Siccome il primo blocco P_1 non ha un predecessore, viene messo in XOR con il blocco iniziale $C_0 = \text{IV}$. L'IV deve essere composto da byte casuali e impredicibili (ad es. non in sequenza), altrimenti l'attaccante potrebbe usare i database relativi ad IV "facili" per compiere l'attacco known-plaintext sul primo blocco in breve tempo. L'IV può essere inviato:

- in chiaro: deve essere cambiato a ogni trasmissione, altrimenti l'attaccante avrebbe il tempo per costruire il database relativo a quel particolare IV;
- criptato con la modalità ECB (1 blocco): l'IV può essere di lunga durata.

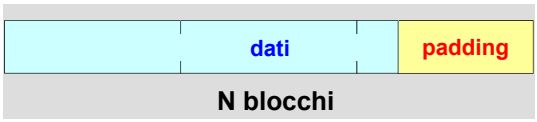
Vantaggi

- no attacchi di swapping: lo scambio di due blocchi C_i e C_{i+1} produrrà un output decriptato non comprensibile ai blocchi P_i, P_{i+1} e P_{i+2};
- no attacchi known-plaintext: blocchi identici non sono criptati allo stesso modo.

Svantaggi

- propagazione degli errori: un errore nella trasmissione al blocco C_i provoca un errore nella decriptazione ai blocchi P_i e P_{i+1};
- no parallelizzazione della CPU: le operazioni crittografiche sono sequenziali.

2.4.3 Padding



La tecnica del **padding** (o allineamento, o riempimento) consiste nell'aggiungere L bit in fondo all'ultimo N-esimo blocco per rendere la dimensione D dei dati un multiplo del blocco base B, prima di poter applicare un algoritmo crittografico in modalità ECB o CBC:

$$D + L = N \cdot B, \quad 1 \leq L \leq B$$

Se la dimensione dei dati è un multiplo esatto del blocco base, occorre comunque inserire un blocco intero di padding in modo da evitare errori di interpretazione dell'ultimo blocco di dati ($L = B$).

Metodi di padding

Quali valori devono avere i bit di riempimento affinché sia possibile distinguere il padding dai dati effettivi?

- tutti i byte con valore null: `...0x00 0x00 0x00` (se la lunghezza dei dati è nota a priori o è ricavabile, ad es. terminatore di stringa `\0`);
- un bit a 1 seguito da bit a 0: `...1000000` (DES originale);
- un byte con valore 128 seguito da byte con valore null: `...0x80 0x00 0x00`;
- ultimo byte con valore L: `...0x03`:
  - preceduto da byte con valore null: `...0x00 0x00 0x03` (Schneier);
  - preceduto da byte con valori casuali: `...0x05 0xf2 0x03` (SSH2);
  - preceduto da byte con valore L: `...0x03 0x03 0x03` (SSL/TLS, PKCS #5, PKCS #7);
  - sequenza di numeri progressivi: `...0x01 0x02 0x03` (IPsec, ESP);
- tutti i byte con valore $L - 1$: `...0x02 0x02 0x02`.

Osservazioni

- La scelta del tipo di padding per un certo algoritmo determina il tipo di alcuni attacchi possibili: ad esempio, siccome SSH2 usa padding casuale, dati uguali vengono criptati in testi cifrati differenti, così non si può compiere un padding oracle attack.
- I tipi di padding con lunghezza del padding esplicita offrono anche un controllo (minimale) di integrità: un'alterazione dei dati nella trasmissione si può propagare ai byte di padding.

2.4.4 CTS

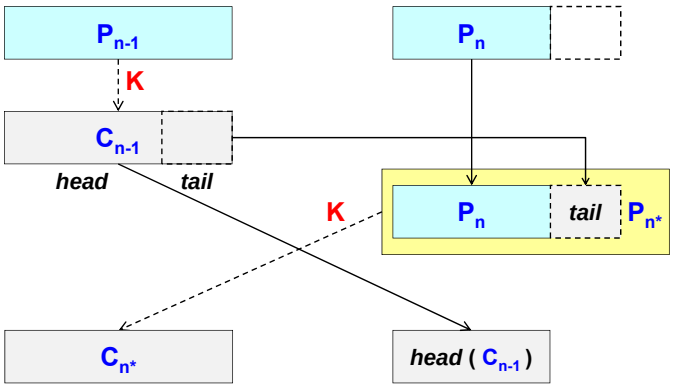


Figura 2.5: Criptazione in modalità ECB con tecnica CTS.

La tecnica del **ciphertext stealing** (CTS) permette di usare gli algoritmi a blocchi in modalità ECB o CBC senza ricorrere al padding:

1. il penultimo blocco P_{n-1} viene criptato usando la chiave K, ottenendo il blocco cifrato C_{n-1};
2. il blocco C_{n-1} viene suddiviso in due parti: una testa e una coda, quest'ultima esattamente della dimensione che manca all'ultimo blocco P_n per arrivare alla dimensione del blocco base;
3. la coda del blocco C_{n-1} viene spostata in fondo all'ultimo blocco P_n, ottenendo il blocco P_{n^*} che ora è un multiplo del blocco base;
4. P_{n^*} viene criptato usando la chiave K, ottenendo il blocco C_{n^*};
5. la testa del blocco C_{n-1} e il blocco C_{n^*} vengono scambiati, perché nella decriptazione bisogna ottenere la coda prima di poter decriptare il blocco C_{n-1}.

Vantaggio La dimensione del testo cifrato C è uguale alla dimensione del testo in chiaro P: una parte dei dati stessi è usata come padding, evitando l'overhead del padding $\Rightarrow$ ciò è utile per la criptazione di settori di disco: non è possibile superare lo spazio su disco.

Svantaggio Il tempo di elaborazione aumenta a causa delle operazioni da effettuare sull'ultimo ed il penultimo blocco, in fase sia di criptazione sia di decriptazione.

2.4.5 CFB

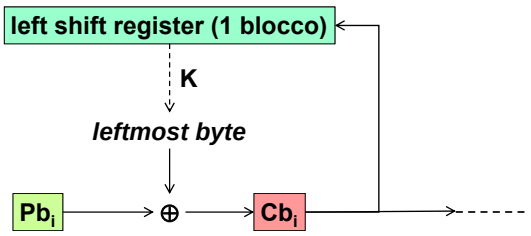


Figura 2.6: Criptazione in modalità CFB applicata al gruppo i-esimo di dimensione $N = 8$ bit.

La modalità **Cipher FeedBack** (CFB) permette di criptare un gruppo di N bit alla volta, con N variabile e inferiore alla dimensione B del blocco base:

- il blocco memorizzato in un registro a scorrimento verso sinistra viene criptato usando la chiave K;
- dal risultato della criptazione, vengono presi tanti bit più a sinistra quanti sono i bit nel gruppo Pb_i da criptare in arrivo dalla trasmissione;
- questi bit vengono messi in XOR con il gruppo Pb_i, ottenendo il gruppo cifrato Cb_i;
- il gruppo Cb_i viene inviato al destinatario come parte del testo cifrato;
- il gruppo Cb_i viene inserito nella parte destra del registro a scorrimento.

È richiesto un IV per impostare il registro a scorrimento prima di poter criptare il primo gruppo.

Svantaggio Un errore di trasmissione su un gruppo causerà un errore nella decriptazione di un intero blocco (a partire dal gruppo Cb_i compreso), fino a quando l'errore non esce dal registro a scorrimento.

2.4.6 OFB

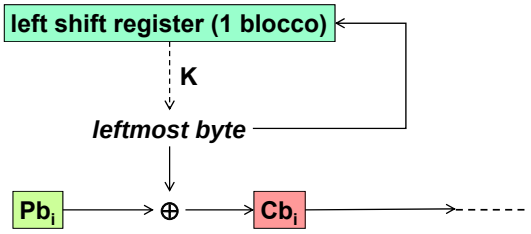


Figura 2.7: Criptazione in modalità OFB applicata al gruppo i-esimo di dimensione $N = 8$ bit.

La modalità **Output FeedBack** (OFB) si differenzia dalla modalità CFB per il fatto che la reazione è alimentata non dal gruppo cifrato Cb_i, ma dal gruppo che viene messo in XOR.

Vantaggio Un errore di trasmissione su un gruppo causerà un errore nella decriptazione solo di quel gruppo.

Svantaggio Siccome la reazione non prende mai dati dall'esterno, i valori nel registro a scorrimento sono ripetitivi e a lungo andare prevedibili.

2.4.7 CTR

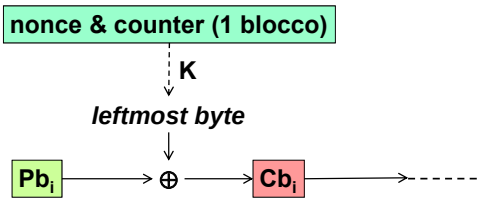


Figura 2.8: Criptazione in modalità CTR applicata al gruppo i-esimo di dimensione $N = 8$ bit.

liveness una proprietà di un'associazione comunicativa o una funzionalità di un protocollo di comunicazione che fornisce la garanzia al destinatario dei dati che i dati stanno venendo trasmessi in modo fresco dal suo originatore, ad es. che i dati non stanno venendo ripetuti,

da parte dell'originatore o di un terzo, da una trasmissione precedente

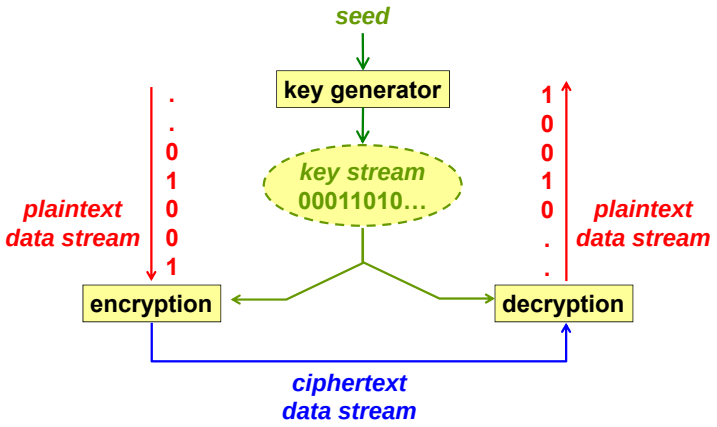
nonce un valore casuale o che non si ripete che è incluso nei dati scambiati da un protocollo, di solito con lo scopo di garantire la liveness e quindi di rilevare e proteggere da attacchi replay

Il **Counter mode** (CTR) si differenzia dalle due modalità precedenti per il fatto che il registro non è più a scorrimento, ma memorizza la combinazione (concatenazione, somma, XOR...) tra un contatore e un nonce (= numero usato una volta).

Vantaggi

- il contatore cambia il valore del registro a ogni gruppo in modo imprevedibile;
- il nonce cambia il valore del registro a ogni trasmissione in modo imprevedibile (il nonce viene trasmesso come IV);
- un errore di trasmissione su un gruppo causerà un errore nella decriptazione solo di quel gruppo (come la modalità OFB);
- parallelizzazione della CPU: un qualsiasi gruppo può essere criptato senza dover criptare i gruppi precedenti (accesso casuale).

2.5 Algoritmi simmetrici di tipo stream



cifrario di tipo stream un algoritmo di criptazione che rompe il testo in chiaro in un flusso di elementi successivi (solitamente bit) e cripta l'n-esimo elemento del testo in chiaro con l'n-esimo elemento di un flusso parallelo di chiave, convertendo così il flusso in chiaro in un flusso cifrato

one-time pad un algoritmo di criptazione in cui la chiave è una sequenza casuale di simboli e ogni simbolo viene usato per la criptazione una sola volta – ad es. usato per criptare un solo simbolo in chiaro e produrre così un solo simbolo cifrato – e una copia della chiave viene usata similmente per la decriptazione

pseudocasuale una sequenza di valori che sembra casuale (ad es. impredicibile) ma è in realtà generata da un algoritmo deterministico

seme un valore che è un ingresso a un generatore di numeri pseudocasuali

Un **algoritmo simmetrico di tipo stream** opera su un flusso di dati senza richiederne la suddivisione in blocchi: a ogni bit del flusso di dati in chiaro corrisponde un bit del flusso di

chiave, e questi bit sono combinati insieme in qualche modo (tipicamente tramite l'operatore XOR) per ottenere il flusso di dati cifrati.

Il flusso di chiave deve essere lo stesso in fase sia di criptazione sia di decriptazione:

- one-time pad: idealmente la chiave deve essere veramente casuale, ma la chiave è lunga quanto il messaggio da criptare ⇒ non è possibile condividere una chiave di tali dimensioni con il destinatario in modo sicuro;
- chiave pseudo-casuale: gli algoritmi reali usano generatori pseudo-casuali di chiavi, sincronizzati tra mittente e ricevente, e viene condiviso solo il seme da cui è generata la sequenza:
  - il generatore di chiave deve essere pubblico, per il principio di Kerckhoffs;
  - il seme deve essere segreto (condiviso in modo sicuro) e imprevedibile (ad es. non in sequenza).

Vantaggio Gli algoritmi di tipo stream sono adatti per la criptazione di dati all'interno di un computer (ad es. per la criptazione dei backup di dischi):

- trasmissione affidabile: la trasmissione avviene tramite il bus;
- prestazioni: l'algoritmo è estremamente veloce, in quanto le operazioni crittografiche consistono in semplici operazioni di XOR.

Svantaggio Gli algoritmi di tipo stream non sono adatti per la criptazione di dati trasmessi nella rete:

- trasmissione non affidabile: i flussi multimediali viaggiano su UDP, dove i pacchetti possono andare persi ⇒ la perdita di un bit durante la trasmissione (o la cancellazione da parte di un attaccante) causa la desincronizzazione tra il flusso di testo cifrato e il flusso di chiave al lato ricevitore, rendendo quindi impossibile eseguire la decriptazione corretta del messaggio da quel bit in poi;
- pacchettizzazione: poiché i dati sono già suddivisi in pacchetti, sono più adatti gli algoritmi a blocchi.

2.6 Principali algoritmi simmetrici

	lunghezza chiave	**dimensione blocco**	**note**	**sez.**
DES	56 bit	64 bit	obsoleto	2.6.1
3DES	112 bit	64 bit	2 chiavi, resistenza 56/112 bit	2.6.2
	168 bit	64 bit	3 chiavi, resistenza 112 bit	
IDEA	128 bit	64 bit		2.6.3
RC2	da 8 a 1024 bit	64 bit	solitamente $K = 64$ bit	2.6.4
RC4	variabile	stream	segreto	
RC5	da 0 a 2048 bit	da 1 a 256 bit	ottimale se $B = 2W$	2.6.5
AES	128, 192 o 256 bit	128 bit	alias Rjindael	2.6.6

2.6.1 DES

L'algoritmo **Data Encryption Standard** (DES) fu progettato negli anni '70 ed è stato lo standard del governo federale americano fino al 1999.

Il DES è pensato per essere efficiente in hardware, perché richiede solo primitive implementate in ogni CPU:

- XOR;
- shift;
- permutazione (un colpo di clock è sufficiente per eseguire tutte le permutazioni necessarie).

Siccome, da quando l'algoritmo fu inventato, le capacità di calcolo dei processori sono cresciute enormemente, l'attacco a forza bruta è diventato sempre più fattibile per i moderni processori: il DES non è un algoritmo intrinsecamente debole, ma il problema è che la chiave utilizzata è troppo corta: la chiave effettiva è lunga complessivamente 64 bit, ma la chiave efficace è lunga solo 56 bit in quanto è inserito un bit di parità ogni 7 bit $\Rightarrow$ la resistenza T all'attacco a forza bruta è pari a 2^{56}.

Alla fine degli anni '90 il crittografo Ron Rivest, per dimostrare la vulnerabilità di DES a favore dei suoi algoritmi, diede vita ad una serie di competizioni con l'assegnazione di premi a chi era in grado di decriptare dei messaggi che lui aveva criptato con DES e pubblicato sul Web. Per la DES Challenge III (1998), l'EFF costruì addirittura un sistema special-purpose, chiamato DEEP CRACK, che era in grado di decriptare un generico messaggio DES entro una settimana, anche se con le seguenti limitazioni:

- bisogna conoscere il tipo di dati originali (ad es. ASCII) per riconoscere se l'output è corretto;
- la macchina non riesce a decriptare messaggi 3DES: non bastano 3 DEEP CRACK perché il 3DES è un algoritmo non lineare.

Dopo il 1999 il DES è stato definito come insicuro, e l'IETF cambiò tutti gli RFC sconsigliando l'uso del DES e suggerendo temporaneamente l'uso del 3DES come una soluzione tampone.

2.6.2 3DES

L'algoritmo **Triple DES** (3DES) non è altro che l'applicazione ripetuta del DES.

L'algoritmo solitamente viene usato nella modalità Encrypt-Decrypt-Encrypt (EDE):

$$C' = \text{enc}\,(K_1, P) \Rightarrow C'' = \text{dec}\,(K_2, C') \Rightarrow C = \text{enc}\,(K_3, C'')$$

1. i dati in chiaro sono criptati usando la chiave K_1, ottenendo il testo cifrato C';
2. il testo cifrato C' è decriptato usando la chiave K_2, diversa dalla chiave K_1, ottenendo il testo cifrato C'';
3. il testo cifrato C'' è criptato usando la chiave K_3, diversa dalla chiave K_2, ottenendo il testo cifrato C.

Possono essere usate 2 o 3 chiavi, ciascuna lunga 56 bit:

- $K_1 = K_3$: la chiave efficace K_{eq} sarebbe lunga $56 \cdot 2 = 112$ bit, ma se l'attaccante ha a disposizione 2^{59} byte di memoria, svolgendo un pre-calcolo la resistenza dell'algoritmo scende a 56 bit (come il DES semplice);
- $K_1 \neq K_3$: la chiave efficace K_{eq} è sempre lunga $56 \cdot 3 = 112$ bit.

Il 3DES in modalità EDE, a differenza della modalità EEE, mantiene la compatibilità con il DES: se viene usata una sola chiave ($K_1 = K_2 = K_3$), l'output finale è equivalente all'output del DES semplice $\Rightarrow$ un unico circuito hardware può implementare sia il DES che il 3DES.

2.6.3 IDEA

Il DES era gratuito ma, essendo stato sviluppato anche dai servizi segreti americani (NSA), era sospettato di contenere delle backdoor ⇒ nonostante il DES fosse lo standard del governo federale americano, sono venuti alla luce altri algoritmi di crittografia.

L'**International Data Encryption Algorithm** (IDEA) è un algoritmo brevettato ma con basse royalty (per uso non commerciale era gratuito), di proprietà della ditta svizzera ASCOM AG. È stato famoso fino agli anni 2000 perché era l'algoritmo usato dal software Pretty Good Privacy (PGP).

Risolve il problema principale del DES, cioè la lunghezza della chiave: usa infatti una chiave lunga 128 bit.

IDEA è particolarmente adatto per essere implementato in software, in esecuzione su CPU a 16 bit, perché richiede le seguenti operazioni:

- XOR;
- addizione modulo 16;
- moltiplicazione modulo $2^{16} + 1$ (primitiva delle CPU CISC).

2.6.4 RC2, RC4

Gli algoritmi **RC2** e **RC4** furono sviluppati dal crittografo Ron Rivest, e sono proprietari della sua società RSA ma non sono brevettati. RC2 è stato pubblicato come RFC da Rivest stesso per essere candidato alla competizione per AES, mentre RC4 è attualmente segreto (anche se è stato soggetto a reverse engineering ed esistono delle librerie non ufficiali come ARCFOUR).

RC2 e RC4 sono rispettivamente 3 e 10 volte più veloci di DES; inoltre sono entrambi implementati a livello software. RC2 lavora su blocchi di dati, mentre RC4 è di tipo stream. Entrambi usano chiavi a lunghezza variabile.

2.6.5 RC5

L'algoritmo **RC5**, sviluppato sempre da Rivest, fu pubblicato fin dall'inizio come RFC (Rivest era stato pagato per realizzare un algoritmo appositamente per il Wireless Application Protocol [WAP]).

Molti parametri, come la lunghezza delle chiavi e la dimensione del blocco base, sono variabili, ma l'algoritmo lavora al meglio quando la dimensione del blocco B è uguale al doppio della word della CPU su cui è in esecuzione: $B = 2W$ (**algoritmo adattativo**).

È adatto per essere implementato in hardware e soprattutto in software, perché utilizza le seguenti operazioni:

- shift;
- rotate (primitiva delle CPU CISC, ma si può ottenere con due primitive sulle CPU RISC);
- addizione modulare.

2.6.6 AES

Il fatto che il 3DES applica 3 volte il DES, che si era rivelato insicuro, non suscitava una grande senso di sicurezza ⇒ il governo USA indisse un bando di gara internazionale per trovare un nuovo algoritmo simmetrico, che si sarebbe dovuto chiamare **Advanced Encryption Standard** (AES) e che avrebbe dovuto avere le seguenti caratteristiche:

- lunghezza della chiave almeno 256 bit;
- dimensione del blocco base almeno 128 bit.

Fra 15 algoritmi candidati e 5 finalisti, nel 2000 fu eletto vincitore l'algoritmo Rijndael perché:

- funzionava mediamente abbastanza bene, anche se mai eccellente, in tutti gli ambienti applicativi;
- era l'unico tra i finalisti i cui autori non erano americani $\Rightarrow$ il governo americano voleva dar prova di non aver fatto favoritismi.

Anche se l'AES fu pubblicato nel 2001, è tuttora gradualmente in corso di adozione: prima di adottare un algoritmo di crittografia nuovo, conviene infatti aspettare alcuni anni in modo che sia analizzato a fondo dalla comunità crittografica alla ricerca di possibili attacchi.

2.7 Principali algoritmi asimmetrici

I principali algoritmi di crittografia asimmetrica sono:

- **RSA** (Rivest-Shamir-Adleman): fornisce sia la riservatezza senza segreti condivisi sia l'autenticazione dell'origine dati (sezione 2.7.1);
- **DSA** (Digital Signature Algorithm): fornisce solo l'autenticazione dell'origine dati, perché opera una compressione con perdita che impedisce di ricostruire i dati originali;
- **El-Gamal**: fornisce solo la riservatezza senza segreti condivisi, basandosi sullo stesso problema matematico del DSA;
- **DH** (Diffie-Hellman): è usato principalmente come meccanismo di key agreement (sezione 2.3.1).

2.7.1 RSA

RSA è un algoritmo inventato negli anni '70 da Rivest, Shamir e Adleman. Il brevetto, valido solo negli USA, è scaduto nel 2000.

Generazione delle chiavi

Per generare la chiave asimmetrica, è necessario:

1. scegliere due numeri primi P e Q (grandi), che devono essere segreti, tali che il loro prodotto, il **modulo** N, sia più grande del testo in chiaro p da criptare:
$$N = P \cdot Q < p$$
2. scegliere arbitrariamente l'**esponente pubblico** $E > 1$ tale che:
   - E sia minore della funzione toziente di Eulero φ:
   $$E < \varphi = (P-1)(Q-1)$$
   - E sia relativamente primo[1] rispetto a φ;
3. calcolare l'**esponente privato** D:
$$D = E^{-1} \bmod \varphi$$
4. la chiave simmetrica è costituita dalla coppia chiave privata K_{pri} e chiave pubblica K_{pub}:
$$\begin{cases} K_{\text{pri}} = (N, D) \\ K_{\text{pub}} = (N, E) \end{cases}$$

[1] Un numero è relativamente primo rispetto ad un altro numero se questi non hanno fattori in comune.

I ruoli degli esponenti E e D sono interscambiabili:[2]

- autenticazione dell'origine dati: viene usata la chiave asimmetrica del mittente:
  - firma: il mittente usa la sua chiave privata:
$$c = p^D \bmod N$$
  - verifica della firma: il ricevente usa la chiave pubblica del mittente:
$$p = c^E \bmod N$$
- riservatezza senza segreti condivisi: viene usata la chiave asimmetrica del ricevente:
  - criptazione: il mittente usa la chiave pubblica del ricevente:
$$c = p^E \bmod N$$
  - decriptazione: il ricevente usa la sua chiave privata:
$$p = c^D \bmod N$$

Osservazioni

- L'algoritmo RSA può criptare solo una piccola quantità di dati, precisamente p inferiore al modulo $N \Rightarrow$ RSA è una sorta di algoritmo a blocchi, dove la dimensione del blocco base dipende dal modulo N scelto.
- L'esponente privato D è calcolato come l'inverso dell'esponente pubblico E, che in aritmetica modulare non è un numero razionale (che richiederebbe il supporto ai numeri in virgola mobile), ma è l'insieme dei numeri interi tali che:
$$E^{-1} \bmod \varphi = D_x \Leftrightarrow (D_x \cdot E) \bmod \varphi = 1, \quad D_x \in \{D_1, D_2, \ldots\}$$

Prestazioni

La complessità delle operazioni crittografiche dipende dal numero di bit con valore 1 negli esponenti E e D (quando il bit è a 0, non occorre fare la moltiplicazione):

- chiavi pubbliche: le operazioni di criptazione e di verifica della firma sono solitamente veloci, perché gli esponenti E tipici (3, 17, 65537 [numero di Fermat]) hanno molti bit a 0;
- chiavi private: le operazioni di decriptazione e di firma sono più lente, ma si possono rendere 4 volte più veloci con il teorema cinese dei resti (CRT), usato in PKCS #1, grazie all'equivalenza:[3]
$$f(x) \bmod N \sim f(x) \bmod P \;\&\; f(x) \bmod Q$$

[2] Infatti grazie alla proprietà delle potenze:
$$\left(p^D\right)^E \bmod N = \left(p^E\right)^D \bmod N$$

[3] "&" non è un'operazione AND bit a bit.

Attacchi

RSA è resistente solo se è usato in modo appropriato:

- scelta errata di esponenti pubblici: se viene fornita una firma realizzata con una chiave pubblica il cui esponente E ha molti bit a 1, si provoca un grosso sovraccarico di calcolo nella verifica della firma (attacco DoS):
- scelta errata di esponenti pubblici: se lo stesso messaggio criptato viene inviato a diversi destinatari tutti con $E = 3$, allora chi legge tutti i messaggi può scoprire il testo in chiaro:
  - usare il numero di Fermat ($E = 65537$) invece di 3, come suggerito dallo standard PKCS #1 v1.5;
  - prima di criptare il messaggio, aggiungere sempre del "sale" costituito da padding casuale fresco (= nonce) da almeno 8 byte;
- uso della stessa chiave per firma e criptazione: siccome firma e criptazione sfruttano la stessa operazione (solo invertita), se un attaccante persuade un utente a firmare con la sua chiave privata un messaggio criptato con la sua chiave pubblica, allora ottiene il testo in chiaro originale:
  - usare chiavi RSA diverse per criptazione e firma, per evitare che la firma sia uguale al testo in chiaro;
- uso dell'utente come se fosse un "oracolo": esistono tecniche per trovare il testo in chiaro se un utente si limita a restituire ciecamente la trasformazione RSA dell'input:
  - applicare uno specifico formato all'input prima di criptarlo e firmarlo, e non accettare mai di criptare o firmare dei dati grezzi;
  - se il formato del blocco decriptato è diverso da quello atteso, restituire un generico errore e non il blocco decriptato;
- uso del CRT: è più facile attaccare RSA con delle tecniche di fault injection, ossia delle tecniche che provano a cambiare valori per vedere come cambia il risultato e quali errori vengono generati:
  - se la verifica della firma fallisce, segnalare solo genericamente "firma non valida" senza ulteriori dettagli sull'errore.

2.8 Crittografia a curve ellittiche

> **crittografia a curve ellittiche (ECC)** un tipo di crittografia asimmetrica basata sulla matematica dei gruppi che sono definiti dai punti su una curva, dove la curva è definita da un'equazione quadratica in un campo finito

La **crittografia a curve ellittiche** (ECC) utilizza l'aritmetica delle curve ellittiche bidimensionali, invece dell'aritmetica modulare monodimensionale.

Dal punto di vista concettuale, invece di usare numeri (che sono punti su una retta) si prende lo spazio cartesiano, e i valori che sono validi sono solo i punti che soddisfano l'equazione di una curva ellittica.

Il problema del logaritmo discreto su una curva è più complesso da risolvere per i cattivi rispetto al problema del logaritmo discreto nell'aritmetica modulare $\Rightarrow$ le chiavi per l'ECC possono essere più corte (circa 1/10) per un livello di sicurezza comparabile, riducendo i requisiti di memorizzazione sui dispositivi embedded.

Invece per i buoni tutti gli algoritmi richiedono solo due operazioni elementari che danno come risultato dei punti:

- somme di due punti;

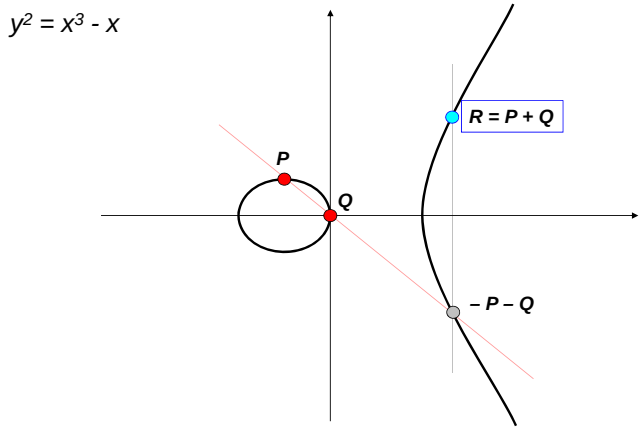


Figura 2.9: Esempio di curva ellittica usata per la crittografia.

- moltiplicazioni di un punto per uno scalare.

Su questo tipo di aritmetica si adattano tutti i problemi visti in precedenza:

- distribuzione delle chiavi: **ECIES** (Elliptic Curve Integrated Encryption Scheme): genera la chiave con operazioni sulla curva;
- firma digitale: **ECDSA** (Elliptic Curve DSA): la firma è una coppia di scalari derivati dal digest e da operazioni sulla curva;
- key agreement: **ECDH** (Elliptic Curve DH): sezione 2.8.1;
- key agreement autenticato: **ECMQV** (Elliptic Curve MQV).

Il governo americano consiglia di usare la crittografia a curve ellittiche per il key agreement e la firma digitale per informazioni "secret" e "top secret".

2.8.1 ECDH

L'**Elliptic Curve Diffie-Hellman** (ECDH) è usato per concordare una chiave segreta K_{AB} in modo analogo al DH classico:

1. X e Y scelgono una stessa curva ellittica ed un suo punto G;
2. X sceglie arbitrariamente un numero x, e calcola $A = x \cdot G$;
3. Y sceglie arbitrariamente un numero y, e calcola $B = y \cdot G$;
4. X e Y si scambiano (pubblicano) i punti A e B;
5. X calcola la chiave $K_A = x \cdot B$;
6. Y calcola la chiave $K_B = y \cdot A$;
7. le chiavi K_A e K_B sono uguali e costituiscono la chiave segreta K_{AB}:

$$K_A = K_B = x \cdot y \cdot G = K_{AB}$$

Anche l'ECDH è resistente agli attacchi di sniffing: l'attaccante sa la curva e i punti G, A e B, ma non sa x e y per calcolare K_{AB}.

2.9 Hashing crittografico

funzione di hash una funzione H che mappa una stringa di bit arbitraria a lunghezza variabile, s, in una stringa a lunghezza fissa, $h = H(s)$ [...]

digest una sorta di "riassunto" a lunghezza fissa dei dati, calcolato solitamente per mezzo di un algoritmo di hash crittografico dedicato (= a scopo di sicurezza)

La criptazione non è una condizione sufficiente per soddisfare il requisito di integrità: una persona che intercetta una comunicazione criptata non può comprendere il significato dei messaggi, ma può modificarli in modo imprevedibile:

- se al lato ricevitore c'è un essere umano, può accorgersi della manipolazione perché i dati decriptati non hanno senso;
- se al lato ricevitore c'è un computer, non sempre può accorgersi della manipolazione perché i dati decriptati sono sempre bit.

Per verificare che i dati ricevuti siano esattamente uguali ai dati inviati, il mittente e il destinatario devono scambiarsi delle informazioni su un canale di comunicazione parallelo (out-of-band):

- manuale: si confrontano dei pezzi scelti a campione ⇒ non si ha la certezza al 100% e sarebbe troppo lungo confrontare tutti i pezzi;
- controllo di ridondanza ciclico (CRC): si confrontano i valori CRC calcolati ⇒ gli algoritmi CRC non sono adatti per la sicurezza: si aspettano che eventi imprevedibili (come il tempo, le radiazioni) danneggino singoli bit equiprobabili in modo casuale, ma un attaccante analizza l'algoritmo e tenta di modificare solo i bit che non sono controllati dall'algoritmo;
- **digest**: si confrontano i "riassunti" calcolati ⇒ sono pensati appositamente per rilevare manipolazioni effettuate da attaccanti.

2.9.1 Algoritmi di hash crittografici

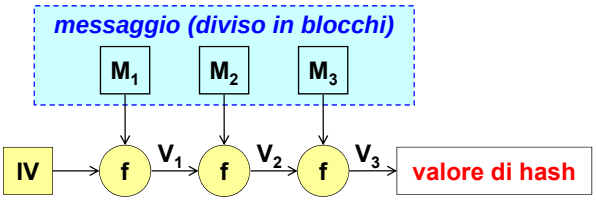


Figura 2.10: Schema generale di una funzione di hash.

Il digest di un messaggio M è calcolato in questo modo:

1. il messaggio M, di qualsiasi lunghezza, viene suddiviso in N blocchi $M_1 \dots M_N$;
2. su ogni blocco M_k, viene applicata iterativamente la **funzione di hash** f, prendendo come input il risultato V_{k-1} del blocco precedente (per il primo blocco è usato un vettore di inizializzazione IV):
$$V_k = f\left(V_{k-1}, M_k\right), \quad V_0 = \text{IV}$$
3. il digest h del messaggio è il risultato V_N dell'ultimo blocco M_N.

Un algoritmo di hash ideale ha le seguenti proprietà:

- prestazioni: il digest è veloce da calcolare;
- unidirezionale: il digest è impossibile da **invertire**, cioè ottenere il messaggio originale dal digest;
- libero da collisioni: è impossibile ottenere una **collisione**, cioè ottenere lo stesso digest da due messaggi originali differenti.

Una volta scelto l'algoritmo, la lunghezza del digest è fissa, indipendentemente dalla lunghezza dei dati in ingresso ⇒ siccome il digest può essere più corto del messaggio originale, il calcolo del digest è un'operazione con perdita, perciò è naturalmente impossibile azzerare la probabilità di **aliasing**: può capitare che due messaggi diversi generino lo stesso digest.

2.9.2 Robustezza di un algoritmo di hash

		lunghezza digest	**dimensione blocco**	**attacchi noti**
MD2		128 bit	8 bit	sì
MD4		128 bit	512 bit	sì
MD5		128 bit	512 bit	sì
SHA-1		160 bit	512 bit	parziali
SHA-2	SHA-224	224 bit	512 bit	no (?)
	SHA-256	256 bit		
	SHA-384	384 bit	1024 bit	
	SHA-512	512 bit		
SHA-3	SHA3-224	224 bit	1152 bit	no
	SHA3-256	256 bit	1088 bit	
	SHA3-384	384 bit	832 bit	
	SHA3-512	512 bit	576 bit	
RIPEMD		160 bit	512 bit	no

Tabella 2.1: Principali algoritmi di hash.

Un attaccante può sfruttare le collisioni per modificare i dati trasmessi senza che il ricevitore se ne accorga: per esempio, in una trasmissione attraverso la rete in cui ogni pacchetto è protetto usando un digest, è possibile compiere l'**attacco del compleanno**[4]: l'attaccante aspetta che vengano generati due messaggi aventi lo stesso digest (collisione), e poi li scambia di posto.

Se l'algoritmo è ben progettato, allora la sua robustezza dipende dalla lunghezza del digest:

- dati due input diversi qualsiasi, la **probabilità di aliasing** P_A è inversamente proporzionale a 2^N, con N = numero di bit del digest:
$$P_A \propto \frac{1}{2^N}$$
Questo dato puramente statistico assume però che vengano modificati dei bit casuali (**rumore bianco**), mentre un attaccante modifica dei bit specifici;

[4] Il nome di questo attacco deriva dal **paradosso del compleanno**: se in una stanza ci sono almeno 23 persone, allora la probabilità che due di loro siano nate nello stesso giorno è maggiore del 50%.

- l'unico attacco possibile è l'attacco del compleanno, che è infattibile in un tempo ragionevole se il digest è sufficientemente lungo: quando il numero di messaggi generati arriva a $2^{\frac{N}{2}}$, allora si ha circa il 50% di probabilità di trovare almeno due messaggi diversi che hanno lo stesso digest.

Un algoritmo di hash, che appare ottimo appena inventato, può diventare obsoleto a causa di attacchi scoperti nel corso del tempo che sfruttano debolezze intrinseche legate alla progettazione dell'algoritmo. Se viene scoperto un attacco che permette di avere una probabilità di aliasing del 50% con un numero minore di input rispetto a quelli necessari per l'attacco del compleanno, riducendo quindi il tempo per trovare una collisione, allora l'algoritmo di hash non è più considerato sicuro:

- **MD2**, **MD4**, **MD5**: sono attualmente considerati insicuri per la scoperta di attacchi definitivi;
- **SHA-1**: la sua robustezza è attualmente messa in dubbio per la scoperta di attacchi parziali;
- **SHA-2**: è una famiglia di algoritmi uguali a SHA-1 ma con maggiori lunghezze del digest $\Rightarrow$ siccome gli errori progettuali di SHA-1 rimangono, gli attacchi per SHA-1 teoricamente sono validi anche per SHA-2, anche se non sono attualmente fattibili in tempi ragionevoli;
- **SHA-3** (alias Keccak): scelto con una competizione internazionale per la progettazione elegante, la facilità di analisi e le prestazioni in hardware, è ancora poco utilizzato perché essendo recente (2008) non è ancora stato analizzato a fondo;
- **RIPEMD**: è considerato ottimo perché per il momento non ci sono attacchi noti, ma essendo scarsamente utilizzato non è ancora stato analizzato a fondo (nonostante sia stato pubblicato nel 1996).

criptosistema l'insieme di un algoritmo di crittografia e di un algoritmo di hash usato per garantire la sicurezza di un certo sistema

Un criptosistema è **equilibrato** quando l'algoritmo di crittografia e l'algoritmo di digest hanno la stessa resistenza, cioè quando il numero di bit del digest è il doppio del numero di bit della chiave di criptazione:

- l'algoritmo di hash SHA-256 (digest = 256 bit) è stato sviluppato per essere usato con l'algoritmo di crittografia AES-128 (chiave = 128 bit);
- l'algoritmo di hash SHA-512 (digest = 512 bit) è stato sviluppato per essere usato con l'algoritmo di crittografia AES-256 (chiave = 256 bit).

2.9.3 Autenticazione del digest

Il digest dovrebbe essere inviato su un canale out-of-bound di cui il mittente e il destinatario si fidano, ma non sempre questo canale è disponibile. Se il digest è inviato insieme ai dati, un attaccante potrebbe effettuare dei cambiamenti sui dati e poi ricalcolare il digest per farlo corrispondere ai dati modificati, sostituendolo al digest dei dati originali in modo che la manipolazione non possa essere rilevata.

Il **Message Authentication Code** (MAC), o **Message Integrity Code** (MIC), è un digest protetto in modo che non possa essere modificato da terzi senza che la modifica possa essere rilevata:

- il termine "MIC" è principalmente usato nel campo delle telecomunicazioni, e pone l'accento sul fatto che fornisce integrità del messaggio, compreso il digest stesso;

- il termine "MAC" è principalmente usato nel campo applicativo e dell'informatica, e pone l'accento sul fatto che fornisce autenticazione dei dati del messaggio, compreso il digest stesso.

Per evitare attacchi di tipo replay, viene di solito aggiunto a ogni messaggio un identificativo univoco (non necessariamente progressivo) detto **Message IDentifier** (MID). Il MID deve essere usato sempre associato a un MAC, perché senza garanzia di integrità anche l'ID potrebbe essere modificato.

L'autenticazione del digest può essere basata su:

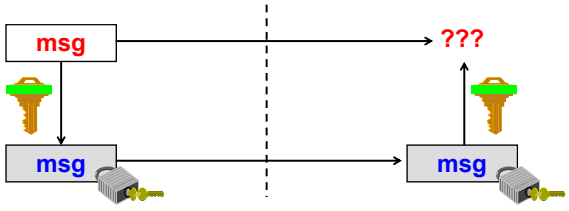
- crittografia simmetrica (sez. 2.10): l'autenticazione è fornita dalla chiave segreta condivisa:
  - essendo veloce si può applicare anche ai dati interi, garantendo la massima integrità;
  - è utile solo per il ricevente: solo lui è sicuro che i dati sono stati generati dal mittente;
  - non fornisce la proprietà di non ripudio: non c'è una prova formale che i dati sono stati generati dal mittente;
- crittografia asimmetrica (sez. 2.11): l'autenticazione è fornita dalla chiave privata del mittente:
  - essendo lenta si può applicare solo al digest, con il rischio di avere collisioni;
  - è utile non solo per il ricevente: tutti sono sicuri che i dati sono stati generati dal mittente;
  - fornisce la proprietà di non ripudio: la firma digitale è la prova formale che i dati sono stati generati dal mittente.

2.10 Autenticazione basata sulla crittografia simmetrica

L'autenticazione è fornita dalla chiave condivisa: essendo una chiave segreta, è conosciuta solo dal mittente e dal destinatario ⇒ solo chi conosce la chiave può modificare il MAC senza che la modifica sia rilevabile: un attaccante non può usare la chiave fintanto che viene tenuta segreta.

A differenza della firma digitale[5], l'autenticazione basata sulla crittografia simmetrica non fornisce la proprietà di non ripudio: siccome la chiave è simmetrica, non è possibile distinguere il ruolo dei due peer, in quanto tutti e due conoscono la stessa chiave condivisa. Questo tipo di autenticazione pertanto non va usata quando i due peer non si fidano l'uno dell'altro: se ci fosse una causa, non sarebbe possibile stabilire chi dei due ha generato il messaggio.

2.10.1 Autenticazione tramite criptazione simmetrica



Il MAC è costituito dalla copia criptata dei dati ⇒ il destinatario potrà decriptare la copia e confrontarla con i dati ricevuti in chiaro.

[5] Si rimanda alla sezione 2.11.

Proprietà di sicurezza

+ integrità: se i dati in chiaro vengono modificati, il risultato della decriptazione della copia sarà diverso dai dati ricevuti;
+ autenticazione: solo chi conosce la chiave segreta può aver generato una copia criptata corrispondente ai dati;
− riservatezza: i dati sono inviati in chiaro;
− non ripudio: non è possibile stabilire quale peer ha generato i dati.

Vantaggio integrità completa: siccome questa tecnica non fa uso di digest, una modifica a qualsiasi bit nel testo in chiaro sarà rilevata al 100% senza il rischio di collisioni

Svantaggi

- quantità di dati trasmessi: gli stessi dati sono inviati due volte;
- tempo di calcolo: il mittente deve criptare tutti i dati e il ricevente deve decriptarli;
- XOR: non si può usare l'algoritmo RC4 come algoritmo di crittografia simmetrica: siccome la criptazione è basata sullo XOR, ogni bit nel testo in chiaro corrisponde a un bit nel testo cifrato alla medesima posizione ⇒ un attaccante può cambiare un bit nei dati originali e il corrispondente bit nei dati cifrati.

2.10.2 Autenticazione tramite CBC-MAC

Il MAC è costituito dall'ultimo blocco di testo cifrato risultante dall'applicazione di un algoritmo di crittografia simmetrica a blocchi in modalità CBC, con vettore di inizializzazione IV nullo (la riservatezza non è necessaria): tutti i blocchi di testo cifrato vengono buttati via eccetto l'ultimo, che viene preso come MAC ⇒ il destinatario potrà criptare i dati ricevuti con lo stesso algoritmo di crittografia e confrontare l'ultimo blocco di testo cifrato risultante con il blocco di testo cifrato ricevuto.

Il **Data Authentication Algorithm** (DAA) usa la tecnica CBC-MAC con l'algoritmo di crittografia DES. Era lo standard del governo USA, ma oggi non è più considerato sicuro a causa delle vulnerabilità di DES.

Proprietà di sicurezza

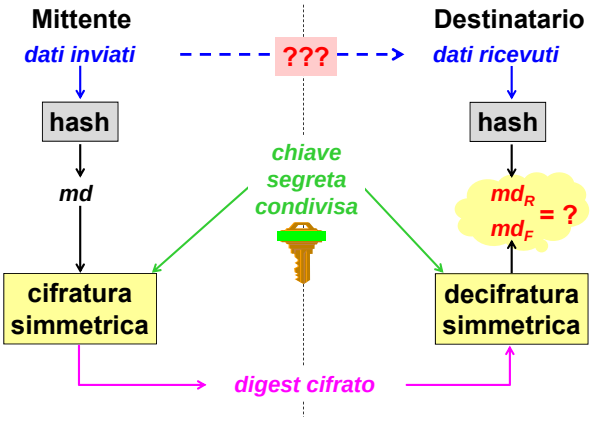
+ integrità: se i dati in chiaro vengono modificati, l'ultimo blocco di testo cifrato risultante dalla criptazione dei dati ricevuti sarà diverso dal blocco di testo cifrato ricevuto;
+ autenticazione: solo chi conosce la chiave segreta può aver generato un blocco di testo cifrato corrispondente ai dati;
− riservatezza: i dati sono inviati in chiaro;
− non ripudio: non è possibile stabilire quale peer ha generato i dati.

Vantaggi

- integrità forte: il blocco finale dipende da tutti i blocchi precedenti ⇒ una modifica molto probabilmente influenzerà l'ultimo blocco;
- implementazione: un singolo algoritmo di crittografia è sufficiente per garantire sia riservatezza sia autenticazione.

Svantaggio attacchi: questo algoritmo è sicuro solo se il messaggio ha una lunghezza fissa o scritta esplicitamente, altrimenti sarebbe possibile effettuare degli attacchi di estensione (per i messaggi a lunghezza variabile occorre usare una variante chiamata **CMAC**)

2.10.3 Autenticazione tramite digest e criptazione simmetrica



Il MAC è costituito dal digest criptato dei dati:

1. il mittente invia dei dati in chiaro al destinatario;
2. il mittente calcola il digest md_F tramite un algoritmo di hash crittografico;
3. il mittente cripta il digest md_F tramite un algoritmo di crittografia simmetrica usando la chiave segreta condivisa;
4. il mittente invia il digest criptato al destinatario;
5. il destinatario calcola il digest md_R sui dati ricevuti tramite lo stesso algoritmo di hash crittografico;
6. il destinatario decripta il digest ricevuto tramite lo stesso algoritmo di hash crittografico usando la chiave segreta condivisa;
7. il destinatario confronta il digest md_R calcolato sui dati ricevuti con il digest md_F ricevuto: se sono uguali, allora i dati saranno integri (a meno di un'eventuale collisione).

Proprietà di sicurezza

+ integrità: se i dati in chiaro vengono modificati, il digest md_R calcolato sui dati ricevuti sarà diverso dal digest md_F ricevuto e decriptato;
+ autenticazione: solo chi conosce la chiave segreta può aver generato un digest criptato corrispondente ai dati;
− riservatezza: i dati sono inviati in chiaro;
− non ripudio: non è possibile stabilire quale peer ha generato i dati.

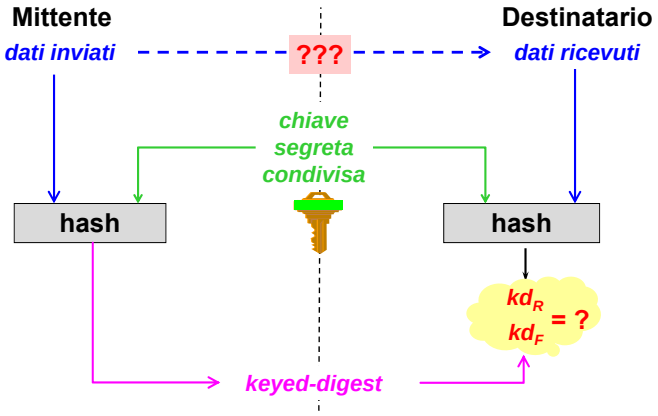
Vantaggi

- quantità di dati trasmessi: il digest (criptato) è molto piccolo rispetto ai dati;
- tempo di calcolo: gli algoritmi di hash sono veloci da eseguire.

Svantaggi

- implementazione: sono necessari due algoritmi diversi, uno di hash e uno di crittografia;
- integrità parziale: possono avvenire delle collisioni, poiché è teoricamente impossibile individuare tutte le possibili modifiche.

2.10.4 Autenticazione tramite keyed-digest



keyed hash un hash crittografico in cui la mappatura a un risultato di hash è variata da un secondo parametro in ingresso che è una chiave crittografica

Il MAC è costituito dal **keyed-digest** dei dati:

1. il mittente invia dei dati in chiaro al destinatario;
2. il mittente calcola il keyed-digest kd_F tramite un algoritmo di hash crittografico usando la chiave segreta condivisa;
3. il mittente invia il keyed-digest kd_F al destinatario;
4. il destinatario calcola il keyed-digest kd_R sui dati ricevuti tramite lo stesso algoritmo di hash crittografico usando la chiave segreta condivisa;
5. il destinatario confronta il keyed-digest kd_R calcolato sui dati ricevuti con il keyed-digest kd_F ricevuto: se sono uguali, allora i dati saranno integri (a meno di un'eventuale collisione).

Proprietà di sicurezza

+ integrità: se i dati in chiaro vengono modificati, il keyed-digest kd_R calcolato sui dati ricevuti sarà diverso dal keyed-digest kd_F ricevuto;
+ autenticazione: solo chi conosce la chiave segreta può aver generato un keyed-digest corrispondente ai dati;
− riservatezza: i dati sono inviati in chiaro;
− non ripudio: non è possibile stabilire quale peer ha generato i dati.

Vantaggi

- quantità di dati trasmessi: il keyed-digest è molto piccolo rispetto ai dati;
- tempo di calcolo: gli algoritmi di hash sono veloci da eseguire;
- implementazione: un singolo algoritmo di hash è sufficiente per garantire sia riservatezza sia autenticazione.

Svantaggio integrità parziale: possono avvenire delle collisioni, poiché è teoricamente impossibile individuare tutte le possibili modifiche

Modi di combinazione

La funzione di hash prende come parametro di input, oltre ai dati, anche la chiave segreta condivisa:

1. la chiave K viene combinata in qualche modo con i dati M:

$$X = g(M, K)$$

2. il digest kd_M viene calcolato applicando la funzione di hash f sul risultato X della combinazione:

$$\mathrm{kd}_M = f(\mathrm{IV}, X) = H(X)$$

La chiave deve essere combinata con i dati in modo appropriato, altrimenti concatenazioni scorrette rendono possibili attacchi che alterano la lunghezza dei dati:

- chiave premessa:

$$\mathrm{kd}_M = f(\mathrm{IV}, K \,||\, M) = H(K \,||\, M)$$

  - il messaggio M in chiaro viene cambiato post-ponendo uno o più blocchi N qualsiasi:

$$M' = M \,||\, N$$

  - il digest kd_M viene sostituito con il digest $\mathrm{kd}_{M'}$ ottenuto applicando la funzione di hash f sul blocco/i N con il digest kd_M come vettore di inizializzazione:

$$\mathrm{kd}_{M'} = f(\mathrm{kd}_M, N) \equiv f(\mathrm{IV}, K \,||\, M \,||N)$$

- chiave posposta:

$$\mathrm{kd}_M = f(\mathrm{IV}, M \,||\, K) = H(M \,||\, K)$$

  - il messaggio M in chiaro viene cambiato premettendo un opportuno blocco N:

$$M' = N \,||\, M, \quad N : \mathrm{IV} = f(\mathrm{IV}, N)$$

  - il digest kd_M non viene toccato:

$$\mathrm{kd}_M = f(\mathrm{IV}, M \,||\, K) \equiv \mathrm{kd}_{M'} = f(\mathrm{IV}, N \,||\, M \,||K)$$

Contromisure

- inserire esplicitamente la lunghezza del messaggio nei dati stessi, in modo che la lunghezza dei dati non sia alterabile;
- aggiungere la chiave sia prima sia dopo il messaggio:

$$\mathrm{kd}_M = H(K \,||\, M \,||\, K)$$

- utilizzare un algoritmo di keyed-digest standard, che usa una specifica funzione di hash applicandola una volta sola:
  - **Keyed MD5** (storico): usa la funzione di hash dell'algoritmo MD5:

$$\mathrm{kd}_M = \mathrm{md5}(K \,||\, \mathrm{keyfill} \,||\, M \,||\, K \,||\, \mathrm{MD5fill})$$

  - **Keyed SHA1** (versione 1 obsoleta): usa la funzione di hash dell'algoritmo SHA1:

$$\mathrm{kd}_M = \mathrm{sha1}(K \,||\, \mathrm{keyfill} \,||\, M \,||\, K \,||\, \mathrm{SHAfill}) \quad \text{(versione 1)}$$

$$\mathrm{kd}_M = \mathrm{sha1}(K \,||\, \mathrm{keyfill} \,||\, M \,||\, \mathrm{datafill} \,||\, K \,||\, \mathrm{sha1fill}) \quad \text{(versione 2)}$$

- usare lo standard **HMAC**, che può usare una qualsiasi funzione di hash H (ad es. MD5, SHA1) applicandola due volte:[6][7]

$$\mathrm{kd}_M = H\left(K' \oplus \mathrm{opad} \,||\, H\left(K' \oplus \mathrm{ipad} \,||\, M\right)\right), \quad \begin{cases} K' = H(K) & |K| > B \\ K' = K & |K| \leq B \end{cases}$$

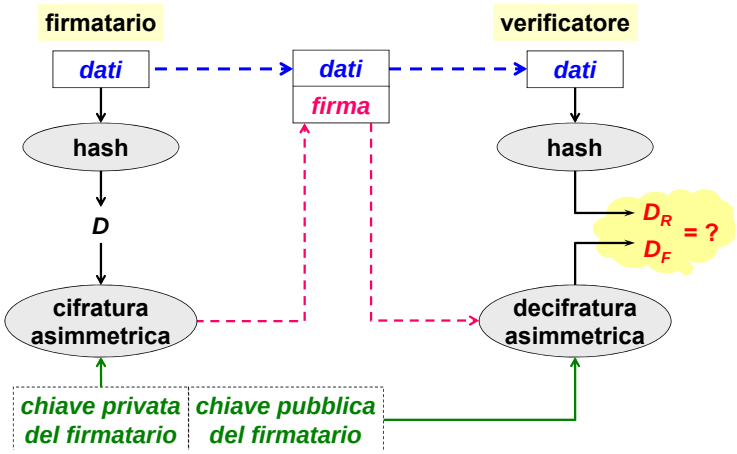
  La funzione di hash viene applicata due volte per rendere il MAC resistente agli attacchi anche se la funzione di hash H non è molto sicura.

2.11 Autenticazione basata sulla crittografia asimmetrica

L'autenticazione è fornita dalla chiave privata del mittente: essendo una chiave segreta, è conosciuta solo dal mittente ⇒ solo chi conosce la chiave può modificare il MAC senza che la modifica sia rilevabile: un attaccante non può usare la chiave fintanto che viene tenuta segreta.

L'autenticazione basata sulla crittografia asimmetrica fornisce la proprietà di non ripudio: siccome la chiave è asimmetrica, è possibile distinguere il ruolo dei due peer, in quanto solo il mittente conosce la sua chiave privata. Tuttavia, il destinatario deve avere la certezza che la chiave pubblica usata sia veramente quella del mittente attraverso un certificato a chiave pubblica (si rimanda alla sezione 3.1).

2.11.1 Firma digitale



firma digitale un valore calcolato con un algoritmo crittografico e associato con un oggetto di dati in modo tale che qualsiasi destinatario dei dati può usare la firma per verificare l'origine e l'integrità dei dati

Firma L'autore di un documento elettronico può firmarlo apponendo la sua **firma digitale**:

1. il firmatario calcola il digest D tramite un algoritmo di hash crittografico;
2. il firmatario cripta il digest D tramite un algoritmo di crittografia asimmetrica usando la sua chiave privata, ottenendo la firma digitale;
3. il firmatario associa la firma digitale ai dati del documento.

[6] La dimensione B del blocco deve essere maggiore della lunghezza L del digest: $B > L$.
[7] L'uso di chiavi K più corte della lunghezza L del digest è deprecato: $|K| \geq L$.

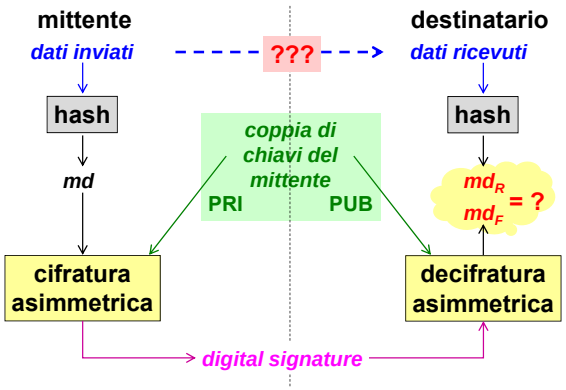
Verifica Chi legge il documento elettronico può verificare se chi sostiene di esserne l'autore lo è veramente tramite la firma digitale allegata:

1. il verificatore calcola il digest D_R tramite lo stesso algoritmo di hash crittografico;
2. il verificatore decripta la firma digitale tramite lo stesso algoritmo di crittografia asimmetrica usando la chiave pubblica dell'autore, ottenendo il digest D_F;
3. il verificatore confronta il digest D_R calcolato con il digest D_F decriptato: se sono uguali, allora i dati saranno integri (a meno di un'eventuale collisione).

La firma digitale è più forte della firma autografa perché è legata indissolubilmente ai dati:

- firma digitale: su un documento elettronico fornisce autenticazione e integrità, perché varia a seconda dei dati (da una singola chiave privata si possono generare infinite firme digitali, una per ogni documento diverso);
- firma autografa: su un documento cartaceo fornisce autenticazione ma non integrità, perché è sempre uguale indipendentemente dai dati ⇒ è possibile aggiungere del testo dopo che il documento è stato firmato.

2.11.2 Autenticazione tramite firma digitale



Il MAC è costituito dalla firma digitale dei dati:

- il firmatario è il mittente;
- il verificatore è il destinatario.

Proprietà di sicurezza

+ integrità: se i dati in chiaro vengono modificati, il digest md_R calcolato sui dati ricevuti sarà diverso dal digest md_F ricevuto e decriptato;
+ autenticazione: solo chi conosce la chiave privata può aver generato una firma digitale corrispondente ai dati;
− riservatezza: i dati sono inviati in chiaro;
+ non ripudio: il mittente non può negare di aver firmato i dati, a patto che la sua chiave pubblica sia fornita da un certificato a chiave pubblica.

2.11.3 PKCS #1

PKCS #1 (Public Key Cryptography Specification #1) è uno standard che implementa le firme digitali, definendo tutte le primitive per usare RSA come algoritmo di crittografia asimmetrica nelle firme digitali:

- conversione e rappresentazione di grandi interi;
- algoritmi base di criptazione/decriptazione;
- algoritmi base di firma/verifica.

Queste primitive devono essere usate in modo opportuno per creare uno "schema crittografico" sicuro standard ⇒ lo standard specifica in modo preciso come eseguire le seguenti operazioni:

- criptazione/decriptazione:
  - RSAES-PKCS1 versione 1.5: vecchio schema base di criptazione che è stato attaccato;
  - RSAES-OAEP (Optimal Asymmetric Encryption Padding): lo schema di criptazione attualmente sicuro in cui viene specificato un padding ottimale, perché una pessima scelta di padding può portare a degli attacchi;
- firma/verifica: (questi schemi sono detti **con appendice** perché ciò che viene criptato non sono i dati ma il loro "riassunto" [digest])
  - RSASSA-PKCS1 versione 1.5: vecchio schema base di firma che è stato attaccato;
  - RSASSA-PSS (Probabilistic Signature Scheme): lo schema di firma attualmente sicuro.

2.11.4 Attacchi alla firma digitale

In caso di attacco, non è possibile individuare l'obiettivo dell'attaccante: se la verifica della firma digitale fallisce, cioè i digest md_R e md_F non sono uguali, ciò può essere dovuto a tre casi tra loro indistinguibili:

- l'attaccante può aver alterato i dati ⇒ il digest md_R non sarà corretto;
- l'attaccante può aver alterato la firma digitale ⇒ il digest md_F non sarà corretto;
- l'attaccante può aver dichiarato di essere il mittente legittimo ⇒ la chiave pubblica del mittente legittimo non corrisponderà alla chiave privata dell'attaccante.
  La chiave pubblica del mittente non deve essere fornita dal mittente, ma deve essere fornita da un'entità esterna fidata attraverso un certificato a chiave pubblica, altrimenti l'attaccante potrebbe spacciare la sua chiave pubblica come la chiave pubblica del mittente legittimo (si rimanda alla sezione 3.1).

La funzione di hash deve avere le seguenti proprietà:

- resistente alle collisioni: dati diversi devono dare origine a firme diverse, altrimenti la stessa firma potrebbe essere usata per altri documenti;
- difficile da invertire: se la funzione di hash fosse invertibile, sarebbe possibile creare delle firme digitali false senza conoscere la chiave privata del mittente:
  1. l'attaccante sceglie una firma digitale S a caso;
  2. l'attaccante cripta la firma digitale S usando la chiave pubblica del mittente legittimo, ottenendo il digest R;
  3. l'attaccante inverte il digest R, ottenendo i dati $X = h^{-1}(R)$;
  4. l'attaccante invia i dati X e la firma digitale S, fingendosi il mittente legittimo;
  5. il ricevente verifica con successo la firma digitale S (falsa) usando la chiave pubblica del mittente legittimo.

2.12 Crittografia con autenticazione (e integrità)

2.12.1 Operazioni separate

I dati possono essere simultaneamente criptati e autenticati effettuando due operazioni separate:

- riservatezza: crittografia simmetrica con la chiave K_1;
- autenticazione (e integrità): MAC (keyed-digest) con la chiave K_2.

Una combinazione scorretta di algoritmi sicuri può condurre a un risultato insicuro:

- **authenticate-and-encrypt** (A&E): il MAC è calcolato sul testo in chiaro ed è inviato in chiaro in parallelo ai dati criptati (ad es. SSH):

$$\text{enc}\left(p, K_1\right) \,||\, \text{mac}\left(p, K_2\right)$$

  - − attacchi DoS: il MAC è calcolato sui dati in chiaro ⇒ occorre sempre decriptare i dati ricevuti prima di poter calcolare il digest da confrontare;
  - − attacchi ad oracolo: il MAC è inviato in chiaro ⇒ il MAC potrebbe fornire delle informazioni indirette sul testo in chiaro;
  - − sicurezza: questa combinazione è insicura, a meno che sia fatta in un passo solo;

- **authenticate-then-encrypt** (AtE): il MAC è calcolato sul testo in chiaro ed è inviato criptato insieme ai dati criptati (ad es. SSL, TLS):

$$\text{enc}\left(p \,||\, \text{mac}\left(p, K_2\right), K_1\right)$$

  - − attacchi DoS: il MAC è calcolato sui dati in chiaro ⇒ occorre sempre decriptare i dati ricevuti prima di poter ottenere entrambi i digest da confrontare;
  - \+ attacchi ad oracolo: il MAC è inviato criptato ⇒ l'attaccante non può fare alcuna deduzione sul testo in chiaro;
  - − sicurezza: questa combinazione è sicura solo con algoritmi simmetrici a blocchi in modalità CBC o con algoritmi simmetrici di tipo stream;

- **encrypt-then-authenticate** (EtA): il MAC è calcolato sul testo criptato ed è inviato in chiaro in parallelo ai dati criptati (ad es. IPsec):

$$\text{enc}\left(p, K_1\right) \,||\, \text{mac}\left(\text{enc}\left(p, K_1\right), K2\right)$$

  - \+ attacchi DoS: il MAC è calcolato sul testo criptato ⇒ si può evitare di decriptare i dati ricevuti se il MAC è errato;
  - \+ sicurezza: questa combinazione è la più sicura, a meno di errori di implementazione (ad es. occorre includere sempre nel MAC il vettore di inizializzazione IV e gli algoritmi di crittografia e di hash usati).

2.12.2 Authenticated encryption

> **meccanismo di authenticated encryption** tecnica crittografica usata per proteggere la riservatezza e garantire l'origine e l'integrità dei dati, e che consiste di due processi componenti: un algoritmo di criptazione e un algoritmo di decriptazione

Gli algoritmi **authenticated encryption** (AE) combinano le operazioni di crittografia e di keyed-digest in una singola operazione:

- \+ riservatezza e autenticazione (e integrità) con un solo algoritmo e una sola chiave;
- \+ maggiore velocità;

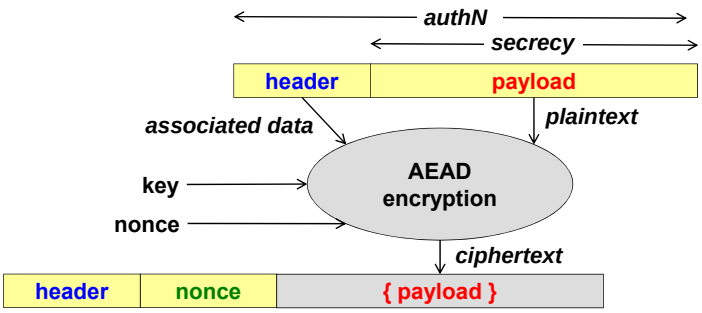
\+ minor rischio di errori di implementazione.

L'AE sta diventando una necessità sempre più crescente a livello applicativo (ad es. per i messaggi di posta elettronica).

Gli algoritmi AE possono essere:

- AEAD o no: alcuni dati autenticati possono essere lasciati non criptati;
- single-pass o double-pass: un algoritmo double-pass ha bisogno di memorizzare i risultati della prima passata per ulteriori calcoli ⇒ è almeno due volte più lento di un algoritmo one-pass se implementato in software;
- on-line o off-line: i dati possono essere presi così come arrivano dal filo o devono essere prima salvati tutti.
  I normali schemi di criptazione sono soggetti ad attacchi chosen-ciphertext (oracle) se usati on-line:
  1. l'attaccante modifica un testo cifrato;
  2. l'attaccante osserva se il ricevente segnala errore o meno.

AEAD



Solo una porzione dei pacchetti di rete necessita di riservatezza:

- necessitano di autenticazione sia l'intestazione sia il payload;
- necessita di riservatezza solo il payload.

Gli algoritmi **Authenticated Encryption with Associated Data** (AEAD) forniscono autenticazione senza riservatezza per dati associati con il testo in chiaro che vanno lasciati non criptati:

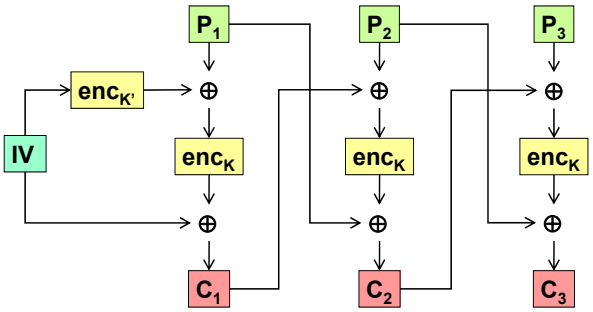
- testo in chiaro: la parte dei dati che necessita anche di riservatezza (ad es. payload);
- dati associati: la parte dei dati che necessita solo di autenticazione (ad es. intestazione).

Il testo cifrato è ottenuto combinando il testo in chiaro e i dati associati insieme ad una chiave, e tipicamente anche ad un nonce per rendere il sistema resistente ad attacchi di tipo replay. Il nuovo pacchetto è formato, oltre che dal testo cifrato, dall'intestazione del pacchetto di partenza e dal nonce usato per la criptazione.

IGE

Infinite Garble Extension (IGE) è una modalità di applicazione degli algoritmi di crittografia simmetrici a blocchi che aggiunge alla modalità CBC la retroazione sul testo in chiaro:

- CBC: un errore al blocco C_i provoca un errore nella decriptazione ai soli blocchi P_i e P_{i+1};



- IGE: un errore al blocco C_i provoca un errore nella decriptazione al blocco P_i e a tutti i blocchi successivi.

IGE è in grado di garantire riservatezza e integrità (e autenticazione) usando un solo algoritmo, ossia l'algoritmo di crittografia: l'integrità è fornita da un blocco di controllo aggiunto alla fine (ad es. tutti zeri, numero di blocchi): se l'ultimo blocco decriptato non è quello previsto, il testo cifrato è stato modificato.

IGE è AE, ma non è AEAD perché non ci sono i dati associati: tutti i dati sono criptati e autenticati.

Altri modi di applicazione per AE

- **GCM** (Galois/Counter Mode) [AEAD, single-pass, on-line]: è il più popolare ed è parallelizzabile (usato da TLS e OpenSSL);
- **OCB** (Offset Codebook Mode) 2.0 [AEAD, single-pass, on-line]: è il più veloce, ma è poco usato perché inizialmente brevettato con licenza GPL;
- **EAX** (Encrypt then Authenticate then X(trans)late) [AEAD, double-pass, on-line]: è lento ma leggero perché usa solo un algoritmo di criptazione ⇒ ottimo per i sistemi limitati:
  - EAX-prime (EAX′): una versione più leggera di EAX che, siccome sacrifica qualche passo di criptazione e qualche byte di memoria, è stata dimostrata essere attaccabile (usata per la trasmissione in rete di misure elettroniche, ad es. contatori Enel);
- **CCM** (CTR mode with CMC-MAC) [double-pass, off-line]: è il più lento (usato nelle reti wireless 802.11i):
  - CCM*: è una versione modificata di CCM che può essere configurato nelle modalità opzionali solo autenticazione o solo criptazione a seconda del contesto operativo (usato da ZingBee, standard di comunicazione per le reti a breve raggio).

È in corso la competizione internazionale CAESAR (2013-2017) per selezionare un algoritmo AE.

2.13 Politica crittografica USA

Fino al 1996, l'esportazione di materiale crittografico, sia in forma software sia in forma hardware, al di fuori dei confini statunitensi era soggetta alle medesime restrizioni del materiale nucleare: i prodotti crittografici (ad es. browser Web) non potevano essere esportati, a meno che il livello di protezione non fosse molto basso (ad es. chiavi simmetriche fino a 40 bit[8], chiavi asimmetriche fino a 512 bit).

Tuttavia le aziende USA stavano perdendo quote di mercato all'estero a causa della scarsa sicurezza offerta dai loro prodotti.

[8]Nel settembre 1998 la lunghezza massima delle chiavi simmetriche fu portato a 56 bit, cioè la lunghezza della chiave efficace dell'algoritmo DES che proprio in quel periodo iniziava a essere considerato non più sicuro.

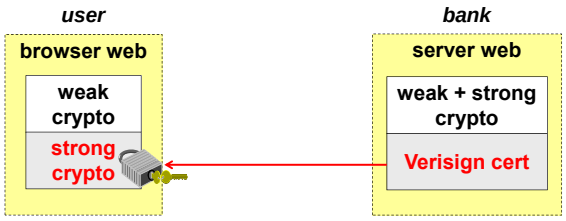
2.13.1 Key escrow (dicembre 1996)

key escrow possibilità di recuperare una chiave anche senza il consenso del proprietario

Le aziende statunitensi furono autorizzate ad esportare prodotti di crittografia semi-robusti (ad es. chiavi simmetriche fino a 56 bit), con la condizione che essi incorporassero delle funzioni di **key escrow**: una chiave poteva essere recuperata in qualche modo dal governo americano se necessario, anche senza il consenso del proprietario.

Esempio Lotus Notes 4.x usava chiavi simmetriche da 64 bit, ma nella versione internazionale 24 di essi erano criptati con la chiave pubblica dei servizi segreti statunitensi (NSA) $\Rightarrow$ il governo americano aveva da indovinare solo 40 bit per recuperare la chiave simmetrica.

2.13.2 Crittografia step-up (gated) (giugno 1997)



Il governo americano concesse il permesso di esportare server Web sicuri purché usati da filiali estere di aziende USA oppure in ambito finanziaro.[9] Il permesso era concesso solo alle entità che acquistavano un certificato a chiave pubblica emesso da VeriSign, che aveva il compito di verificare il reale uso del server Web: quando una banca USA acquistava un certificato di VeriSign, il server Web nella sua filiale estera poteva quindi contenere anche la parte forte di crittografia.

Dalla parte del client, tuttavia, le versioni internazionali dei browser Web sviluppati da aziende statunitensi includevano sia la parte forte sia la parte debole di crittografia:

- siti Web senza certificato VeriSign: il browser usava la parte debole di crittografia, anche se il sito supportava una crittografia più robusta;
- siti Web con certificato VeriSign: il browser riceveva il certificato VeriSign e abilitava (**step up**) la parte forte di crittografia per la navigazione su quello specifico sito Web.

2.13.3 Key recovery

key recovery meccanismi e processi che consentono a parti autorizzate di recuperare la chiave crittografica usata per la riservatezza dei dati

Successivamente gli USA introdussero il permesso di esportare prodotti con crittografia forte (ad es. 128 bit per le chiavi simmetriche), a patto che incorporassero delle funzioni di **key recovery**: quando una chiave veniva generata, essa veniva depositata in uno dei quattro centri autorizzati dal governo USA, criptata con la chiave pubblica del centro di recovery:

+ l'utente poteva recuperare la chiave in caso di smarrimento;
− il governo americano poteva recuperare la chiave per esempio in caso di sospetto terrorismo.

[9] Nel settembre 1998 il permesso fu esteso ad assicurazioni ed istituzioni sanitarie.

2.13.4 Liberalizzazione (parziale) (gennaio 2000)

I prodotti statunitensi potevano finalmente essere esportati off-the-shelf, cioè senza la necessità di autorizzazioni, a patto che fosse rispettata una delle seguenti condizioni:

- il codice sorgente è liberamente disponibile su Internet (open source), oppure
- il prodotto ha superato una "one-time review" (si sospetta che l'NSA inserisca delle backdoor nel codice sorgente).

Capitolo 3

Lo standard X.509, le PKI ed i documenti elettronici

3.1 Certificato a chiave pubblica

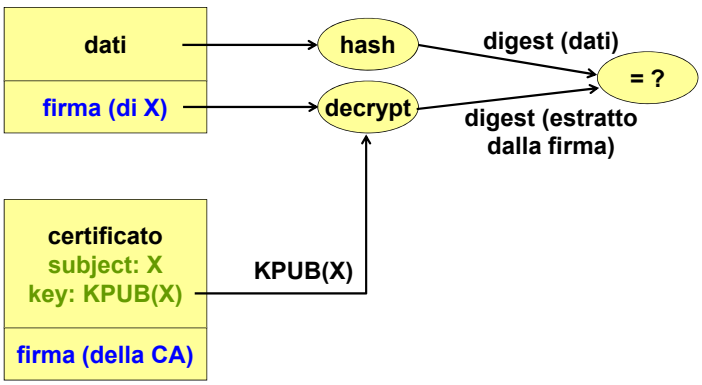


Figura 3.1: Verifica di una firma digitale per mezzo di un certificato a chiave pubblica.

certificato a chiave pubblica una struttura dati usata per legare in modo sicuro una chiave pubblica ad alcuni attributi

I **certificati a chiave pubblica** permettono di distribuire le chiavi pubbliche per usare le firme digitali in modo sicuro: la chiave pubblica del mittente non è fornita dal mittente, ma è fornita da un'entità esterna fidata attraverso un certificato a chiave pubblica, altrimenti un attaccante potrebbe spacciare la sua chiave pubblica come la chiave pubblica del mittente legittimo.

Solitamente un certificato lega una chiave pubblica ad un'identità, cioè a una persona fisica, ma gli attributi dipendono dal contesto applicativo:

- sicurezza delle reti: gli attributi sono gli indirizzi IP;
- sicurezza della posta elettronica: gli attributi sono gli indirizzi di posta elettronica;
- sicurezza del Web: gli attributi sono gli URL.

Formati per certificati a chiave pubblica

- PKCS #6: il vecchio standard dell'RSA, reso obsoleto dalla versione 3 di X.509;
- X.509 (sez. 3.2): è lo standard attualmente più usato:

  - versioni 1 (1988) e 2 (1993): sono state definite solo dall'ISO come parte integrante del sistema OSI, ma non sono molto soddisfacenti;
  - versione 3 (1996): è stata definita dall'ISO congiuntamente con l'IETF, ed è adatta per le necessità di sicurezza di Internet grazie all'introduzione delle estensioni e dei certificati di attributo;

- altri (ad es. PGP[1], SPKI): soluzioni alternative, create da soggetti che rifiutano l'utilizzo di X.509, che però non hanno quasi nessun utilizzo pratico.

3.1.1 PKI

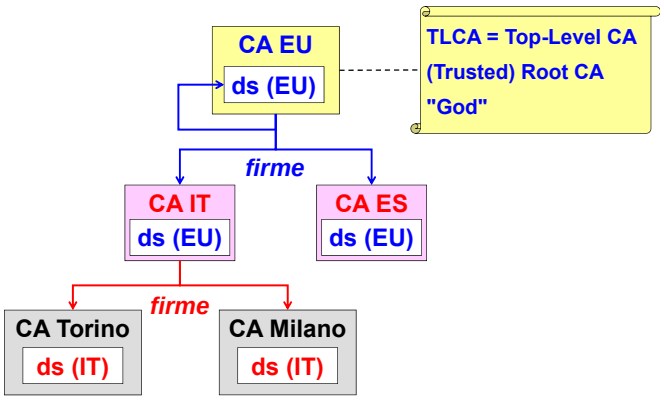


Figura 3.2: Gerarchia di certificazione.

public-key infrastructure (PKI) l'infrastruttura tecnica ed organizzativa preposta alla creazione, distribuzione e revoca dei certificati a chiave pubblica

certification authority (CA) un'entità che emette certificati digitali (soprattutto certificati X.509) e attesta la corrispondenza tra gli elementi di dato in un certificato

registration authority (RA) un'entità PKI facoltativa (separata dalle CA) che non firma né certificati digitali né CRL ma ha la responsabilità di registrare o verificare alcune o tutte le informazioni (in particolare le identità dei soggetti) necessarie a una CA per emettere i certificati e le CRL e per svolgere altre funzioni di gestione dei certificati

end entity un'entità di sistema che è il soggetto di un certificato a chiave pubblica e che sta usando, o è autorizzato e in grado di usare, la chiave privata corrispondente solo per scopi diversi dalla firma di un certificato digitale, cioè un'entità che non è una CA

relying party un'entità di sistema che dipende dalla validità delle informazioni (come il valore della chiave pubblica di un'altra entità) fornite da un certificato digitale

gerarchia di certificazione una topologia strutturata ad albero (priva di cicli) di relazioni tra le CA e le entità a cui le CA emettono certificati a chiave pubblica

CA radice la CA che è la CA di livello più alto (la più fidata) in una gerarchia di certificazione, cioè l'autorità sulla cui chiave pubblica tutti gli utenti dei certificati basano la validazione di certificati, CRL, percorsi di certificazione e altri costrutti

[1]Si rimanda alla sezione 9.7.1.

certificato self-signed un certificato a chiave pubblica per cui la chiave pubblica associata dal certificato e la chiave privata usata per firmare il certificato sono componenti della stessa coppia di chiavi, che appartiene al firmatario

La **Public-Key Infrastructure** (PKI) è composta da entità quali:

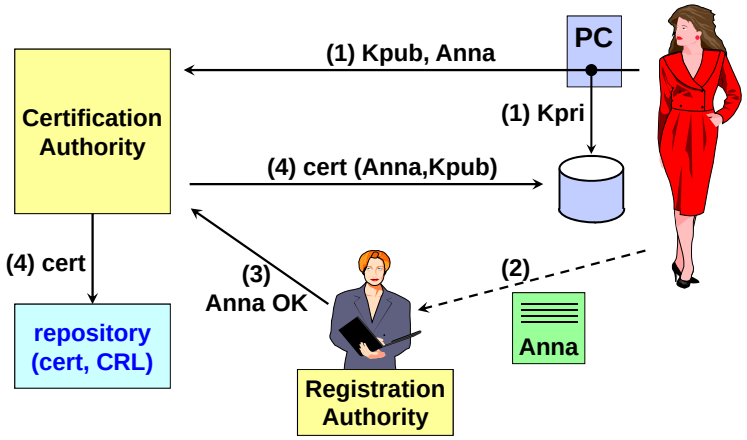
- **Certification Authority** (CA): è la componente tecnica e ha il compito di emettere i certificati a chiave pubblica;
- **Registration Authority** (RA): è la componente organizzativa e ha il compito di verificare l'identità di chi richiede un certificato.

La chiave è legata in modo sicuro grazie al fatto che il certificato è firmato in modo elettronico dall'emettitore, cioè la CA, che è un'entità esterna fidata ⇒ la firma digitale della CA allegata al certificato fornisce integrità e autenticazione, impedendo la creazione di certificati falsi. Con questo sistema, il certificato che viene usato per verificare una firma digitale contiene la firma digitale della CA, che andrebbe verificata con un altro certificato a sua volta contenente la firma digitale di un'altra CA, e così via all'infinito.

Le CA sono organizzate in una **gerarchia di certificazione**: le firme digitali contenute nei certificati di una CA sono verificate da certificati della CA al livello superiore. In cima alla gerarchia di certificazione si trova una **top-level CA**, la quale emette dei **certificati self-signed**, ovvero firmati da se stessa. Sono considerati validi i certificati self-signed solo delle CA incluse in un elenco di top-level CA attendibili; questo elenco è memorizzato sul dispositivo dell'utente e preconfigurato dal produttore (altre top-level CA possono essere aggiunte dall'utente).

TSL La **Trust service Status List** (TSL) è una lista firmata pubblicata dall'Unione Europea per comunicare i Trust-Service Provider (TSP), cioè le autorità fidate (ad es. CA, server OCSP, Time-Stamping Authority), di ogni Stato membro.

3.1.2 Emissione dei certificati



emissione generare e firmare un certificato digitale (o una CRL) e, solitamente, distribuirlo e renderlo disponibile a potenziali utenti del certificato (o utenti della CRL)

soggetto il nome (di un'entità di sistema) che è associato agli elementi di dato in un certificato digitale; ad es., un DN che è associato a una chiave nel certificato a chiave pubblica

emettitore la CA che firma un certificato digitale o una CRL

repository un sistema per archiviare e distribuire i certificati digitali e le relative informazioni (compresi le CRL, i CPS e le politiche di certificato) agli utenti dei certificati

In generale un certificato può anche essere creato direttamente dall'utente, ma un attaccante potrebbe dichiarare di possedere la chiave privata corrispondente alla chiave pubblica di qualcun altro ⇒ la CA deve certificare la corrispondenza tra la chiave pubblica e l'identità del possessore della chiave privata corrispondente.

Un certificato a chiave pubblica viene generato nel seguente modo:

1. l'utente genera la coppia di chiavi che costituisce la sua chiave asimmetrica:
   - la chiave privata viene memorizzata localmente sul dispositivo dell'utente, ed è compito dell'utente conservarla in modo che rimanga segreta;
   - la chiave pubblica viene inviata alla CA attraverso la rete, insieme all'identità dell'utente (ad es. con una richiesta PKCS #10: si rimanda alla sezione 3.9);
2. l'utente si reca personalmente presso una RA e consegna un documento che attesta la sua identità;
3. la RA comunica alla CA che la richiesta di certificato può essere accettata;
4. la CA emette ed invia all'utente un certificato a chiave pubblica contenente l'associazione tra l'identità dell'utente e la chiave pubblica;
5. la CA pubblica il nuovo certificato in un **repository** (= deposito) pubblico, da cui è possibile recuperare tutti i certificati emessi dalla CA (comprese le informazioni di revoca, ad es. CRL).

3.1.3 Revoca dei certificati

revoca di certificato l'evento che accade quando una CA dichiara che un certificato digitale precedentemente valido emesso da quella CA è diventato non valido; solitamente dichiarato con una data effettiva

Il certificato ha una scadenza temporale, e quindi va rinnovato periodicamente. Esso è revocabile prima della sua scadenza naturale:

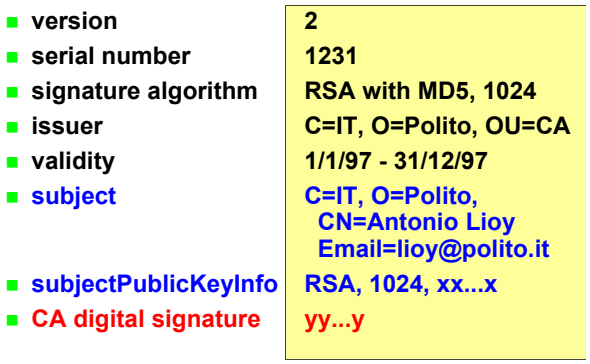
- su richiesta del titolare della chiave privata (**soggetto**): può chiedere la revoca del certificato se non ha più il controllo della sua chiave privata (ad es. è stata rubata);
- autonomamente dalla CA (**emettitore**): può chiedere la revoca del certificato se scopre di essere stata truffata (ad es. l'utente ha dichiarato di essere qualcun altro).

Quando si valida una firma digitale, è compito del ricevente (**relying party**) verificare se il certificato era valido o era stato revocato all'atto della firma, tramite due meccanismi possibili:

- CRL (sez. 3.3): l'utente scarica l'elenco, firmato dalla CA o da un delegato (ad es. RA), contenente tutti i certificati revocati con le relative date di revoca:
  - \+ verifica ritardata: è possibile verificare, anche off-line, la validità di un certificato in una qualsiasi data passata (ad es. all'atto della firma);
  - − risposta grande: è necessario scaricare un grosso elenco;
- OCSP (sez. 3.4): l'utente interroga un server per avere lo stato di un singolo certificato:
  - − verifica in tempo reale: la validità è riferita al momento dell'interrogazione;
  - \+ risposta piccola: il server restituisce solo "good", "revoked" o "unknown".

3.2 Certificati a chiave pubblica X.509

3.2.1 Struttura di un certificato X.509



Lo standard X.509 è descritto usando la sintassi **Abstract Syntax Notation 1** (ASN.1), una notazione che permette di specificare in modo completamente astratto un qualunque formato di dati.

Un certificato X.509 versione 3 contiene, oltre alla firma digitale della CA, una sequenza di 10 elementi di dato, tra cui:

1. `version`: è la versione della codifica del certificato (= versione dello standard X.509 −1);
2. `serialNumber`: è un numero intero assegnato dalla CA, univoco per ogni certificato emesso da una CA data (cioè il nome dell'emettitore e il numero di serie identificano un certificato univoco);
3. `signature`: contiene l'identificativo di algoritmo dell'algoritmo e della funzione di hash usati dalla CA nella firma del certificato (ad es. `sha1WithRSAEncryption`);
4. `issuer`: è il **distinguished name** (DN) che identifica l'entità che ha firmato ed emesso il certificato (ad es. `countryName=IT`, `stateOrProvinceName=Torino`, `organizationName=Politecnico di Torino`, `commonName=CA di prova`);
5. `validity`: è l'intervallo di tempo, compreso tra `notBefore` e `notAfter`, in cui era valida la chiave privata corrispondente alla chiave pubblica che sta venendo certificata;
6. `subject`: è il DN che identifica l'entità associata con la chiave pubblica che sta venendo certificata (ad es. `commonName=Antonio Lioy`);
7. `subjectPublicKeyInfo`: trasporta il valore in chiaro della chiave pubblica che sta venendo certificata, e identifica l'algoritmo di cui questa chiave pubblica è un'istanza (ad es. `rsaEncryption`: esponente pubblico e modulo);

10. `extensions` (facoltativo): contiene eventuali estensioni specifiche del certificato.

3.2.2 Estensioni

Le **estensioni** sono state introdotte dalla versione 3 di X.509 per estendere le definizioni dei certificati e delle CRL, rendendoli più flessibili e accogliendo le diverse necessità.

È responsabilità della CA contrassegnare un'estensione come critica o non critica:

- non critica: se il ricevente non riconosce l'estensione, può scegliere se rifiutare il certificato oppure ignorare l'estensione;

- critica: se il ricevente non riconosce l'estensione, dovrebbe rifiutare l'intero certificato. L'elaborazione è interamente responsabilità del relying party: può decidere di accettare comunque il certificato a suo rischio e pericolo, senza alcuna copertura legale fornita dalla CA.

Esistono estensioni di due tipi:

- standard: estensioni definite nello standard, quindi comprese da tutti i riceventi, suddivise in cinque categorie:
  - **key and policy information**: forniscono informazioni sulla chiave che sta venendo certificata e sulla politica di sicurezza, ovvero le regole formali seguite dalla CA nell'emissione del certificato;
  - **subject and issuer attributes**: forniscono delle informazioni aggiuntive sul soggetto e sull'emettitore;
  - **certification path constraints**: impongono dei vincoli sulla catena di certificazione;
  - **basic CRL extensions**: forniscono informazioni di base sulla CRL;
  - **CRL distribution points and delta-CRLs**: forniscono informazioni sui punti di distribuzione delle CRL e sulle delta CRL;
- private: estensioni definite da una singola azienda o gruppo chiuso, quindi comprese solo da un gruppo ristretto di riceventi (ad es. entro l'azienda).

Estensioni standard

Le estensioni standard specifiche del certificato comprendono:

- key and policy information:
  a) estensione **authority key identifier**: identifica la chiave pubblica da usare per verificare la firma su questo certificato;
  c) estensione **key usage**: identifica lo spazio delle applicazioni crittografiche, cioè gli scopi per cui può essere usata la chiave pubblica (ad es. firma digitale, key agreement);
  d) estensione **extended key usage**: identifica lo spazio delle applicazioni end-user, che possono usare più applicazioni crittografiche (ad es. firma di codice, autenticazione del server);
- subject and issuer attributes:
  a) estensione **subject alternative name**: associa al soggetto del certificato uno o più nomi univoci (ad es. indirizzo di posta elettronica, indirizzo IP);
  b) estensione **issuer alternative name**: associa all'emettitore del certificato uno o più nomi univoci (ad es. indirizzo di posta elettronica, indirizzo IP);
- certification path constraints:
  a) estensione **basic constraints**: indica se il soggetto è una CA o un utente (**end entity**) nell'albero di certificazione, e nel primo caso può specificare la massima profondità del sottoalbero;
  b) estensione **name constraints**: limita il dominio dei nomi fidati che la CA soggetto può inserire nei suoi certificati (ad es. indirizzi di posta elettronica limitati a "*.polito.it");
- CRL distribution points and delta-CRLs:
  a) estensione **CRL distribution points**: contiene le directory entry, gli indirizzi di posta elettronica o gli URL di uno o più punti di distribuzione da cui è possibile reperire le CRL da usare nella verifica della validità del certificato in esame.

Estensioni private

Il gruppo PKIX di IETF ha definito in un RFC tre estensioni private che, date la loro importanza, sono diventate praticamente standard:

- estensione **subject information access**: specifica un metodo (ad es. http) per recuperare le informazioni sul titolare di un certificato e un nome per indicare dove trovarlo (indirizzo);
- estensione **authority information access**: è un puntatore all'indietro ai servizi della CA che ha emesso il certificato (ad es. URL del server OCSP[2]);
- estensione **CA information access**: è un puntatore all'indietro ai servizi della CA che possiede il certificato.

3.3 CRL X.509

certificate revocation list (CRL) una struttura dati che enumera i certificati digitali che sono stati invalidati dal loro emettitore prima di quando si pianificava che scadessero

delta CRL una CRL parziale che contiene solo entry per i certificati che sono stati revocati dall'emissione di una precedente base CRL

indirect certificate revocation list (ICRL) in X.509, una CRL che può contenere le notifiche di revoca di certificato per certificati emessi da CA diverse dall'emettitore (cioè il firmatario) della ICRL

Una **Certificate Revocation List** (CRL) può essere:

- base CRL: contiene l'elenco di tutti i certificati che sono stati revocati prima di una certa data, con le relative date di revoca;
- delta CRL: contiene l'elenco dei soli certificati revocati da quando è stata emessa una precedente base CRL, per fornire **aggiornamenti differenziali** più piccoli e dinamici.

Le CRL sono emesse periodicamente dagli emettitori dei certificati per mantenerle aggiornate. Le CRL possono essere firmate da due entità diverse:

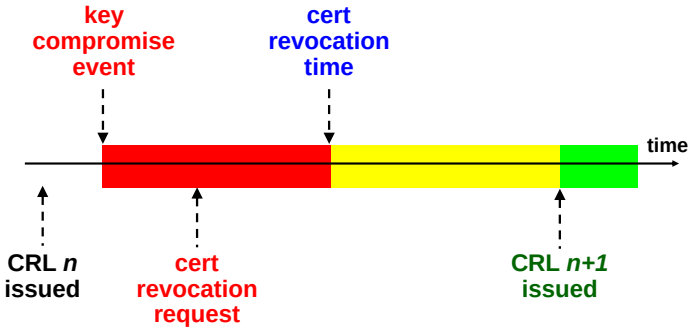
- direttamente dalla CA che ha emesso i certificati revocati;
- da una RA delegata dalla CA (**indirect CRL** [iCRL]).
  Una CA può delegare una RA per motivi organizzativi: la RA può accettare le richieste di revoca ed emettere una nuova CRL anche fuori dagli orari lavorativi degli impiegati della CA.

3.3.1 Sequenza temporale di revoca di un certificato

La revoca di un certificato non è immediata, ma dall'evento di compromissione all'emissione della CRL aggiornata trascorre un certo tempo, durante il quale le firme realizzate con la chiave rubata sono considerate valide:

1. emissione CRL: viene emessa la CRL numero n;
2. evento di compromissione della chiave: la chiave privata dell'utente viene rubata o persa;
3. richiesta di revoca del certificato: l'utente si reca alla CA per chiedere di revocare il certificato associato alla chiave rubata, dichiarando il momento in cui è avvenuto l'evento di compromissione.
   Questa è la **invalidity date**: il titolare della chiave pubblica, non la CA, ha la responsabilità sulla sua veridicità;

[2]Si rimanda alla sezione 3.4.



4. tempo di revoca del certificato: la CA, dopo aver verificato che l'identità del richiedente corrisponda all'identità del soggetto del certificato, revoca il certificato.
   Questa è la **revocation date**: la CA ha la responsabilità sulla sua veridicità;

5. emissione CRL: viene emessa la CRL numero $n + 1$, dove la revoca del certificato viene resa pubblica.

Attacchi La CRL è pressoché inutile se non è aggiornata o se non si sa esattamente quando la firma è stata realizzata:

- attacco back-dating: l'attaccante realizza la firma portando indietro la data del dispositivo, in modo che la firma risulti realizzata prima del furto della chiave ⇒ serve una prova che la firma è stata realizzata prima di un certo istante di tempo (si rimanda alla sezione 3.5);
- attacco DNS spoofing: il campo "issuing distribution point" della CRL non è protetto ⇒ un attaccante può reindirizzare il relying party a una versione vecchia della CRL.

3.3.2 Struttura di una CRL X.509

version	1
signature algorithm	RSA with MD5, 1024
issuer	C=IT, O=Polito, OU=CA
thisUpdate	15/10/2000 17:30:00
userCertificate revocationDate	1496 13/10/2000 15:56:00
userCertificate revocationDate	1574 4/6/1999 23:58:00
CA digital signature	yy...y

Una CRL X.509 versione 2 contiene, oltre alla firma digitale dell'emettitore, una sequenza di 7 elementi di dato:

- `version`: è la versione della codifica della CRL;[3]
- `signature`: contiene l'identificativo di algoritmo dell'algoritmo e della funzione di hash usati dall'emettitore nella firma della CRL (ad es. `sha1WithRSAEncryption`);
- `issuer`: è il DN che identifica l'entità che ha firmato ed emesso la CRL;
- `thisUpdate`: è la data e ora in cui è stata emessa la CRL;

[3]Le designazioni v1 e v2 per una CRL X.509 sono disgiunte dalle designazioni v1 e v2 per un certificato a chiave pubblica X.509, e dalla designazione v1 per un certificato di attributo X.509.

- `nextUpdate` (facoltativo): è la data e ora in cui verrà emessa la prossima CRL;
- `revokedCertificates` (facoltativo): elenca i certificati che sono stati revocati:
  - `serialNumber`: è il numero di serie del certificato revocato;
  - `revocationDate`: è la data e ora di revoca del certificato, cioè quando il titolare della chiave pubblica ha fatto la richiesta di revoca alla CA;
  - `crlEntryExtensions` (facoltativo): contiene eventuali estensioni specifiche della entry (certificato revocato);
- `crlExtensions` (facoltativo): contiene eventuali estensioni specifiche della CRL.

3.3.3 Estensioni

Estensioni specifiche della CRL

Le estensioni standard specifiche della CRL (`crlExtensions`) comprendono:

- key and policy information:
  a) estensione **authority key identifier**: identifica la chiave pubblica da usare per verificare la firma su questa CRL;
- subject and issuer attributes:
  b) estensione **issuer alternative name**: associa all'emettitore della CRL uno o più nomi univoci (ad es. indirizzo di posta elettronica, indirizzo IP);
- basic CRL extensions:
  a) estensione **CRL number**: è il numero di sequenza progressivo della CRL;
- CRL distribution points and delta-CRLs:
  b) estensione **issuing distribution point**: contiene la directory entry, l'indirizzo di posta elettronica o l'URL del punto di distribuzione da cui è possibile reperire la prossima CRL;
  e) estensione **delta CRL indicator**: identifica la CRL come una delta CRL che fornisce aggiornamenti differenziali a una CRL base riferita.

Estensioni specifiche della entry

Le estensioni standard specifiche della entry (`crlEntryExtensions`) comprendono:

- basic CRL extensions:
  b) estensione **reason code**: indica il motivo per cui il certificato è stato revocato (alcuni motivi sono più gravi di altri);
  c) estensione **hold instruction code** (deprecata): permette di sospendere temporaneamente l'uso di un certificato (ad es. un soldato va in vacanza);
  d) estensione **invalidity date**: è la data e ora da cui il certificato non è più valido, cioè quando è avvenuto l'evento di compromissione della chiave;
- CRL distribution points and delta-CRLs:
  d) estensione **certificate issuer**: quando la CRL è indirect, identifica la CA che aveva emesso il certificato revocato e ha delegato la RA.

3.4 OCSP

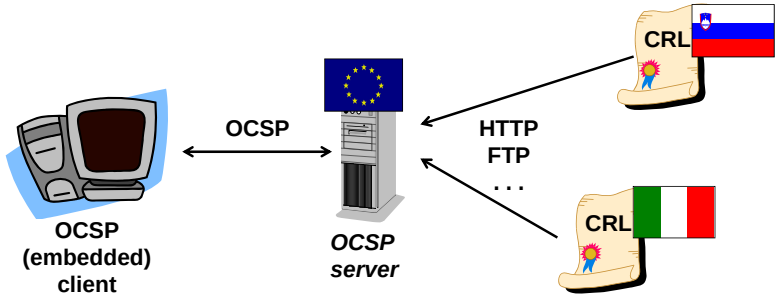


Figura 3.3: Architettura di OCSP.

certificate status responder un server online fidato che agisce per una CA per fornire le informazioni autenticate sullo stato di un certificato agli utenti di certificato; offre un'alternativa all'emissione di una CRL

Lo standard **Online Certificate Status Protocol** (OCSP), definito dal gruppo IETF-PKIX, serve per verificare in linea se un certificato è valido:

1. un **server OCSP** (o OCSP responder) scarica tutte le CRL da cui attingere le informazioni di revoca;
2. il relying party interroga il server OCSP sullo stato corrente del certificato;
3. il server OCSP restituisce lo stato corrente del certificato:
   - good: il certificato non è stato revocato;
   - revoked: il certificato è stato revocato:
     - `revocationTime`: la data e ora di revoca del certificato;
     - `revocationReason`: il motivo della revoca del certificato;
   - unknown: il certificato è sconosciuto.

Le risposte sono firmate dal server OCSP, ma il certificato del server OCSP non è verificabile con OCSP stesso ⇒ spesso i certificati dei server OCSP hanno una durata molto breve (ad es. 24 ore).

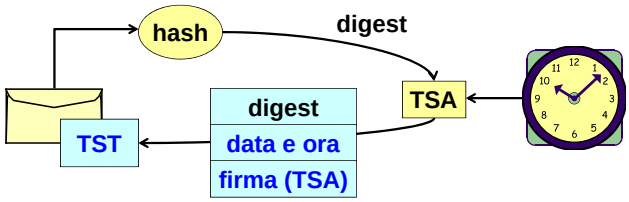
Modelli di server OCSP

- **Trusted Responder**: il server OCSP è pagato dagli utenti:
  - risponde alle richieste su qualsiasi certificato;
  - il certificato del server OCSP è indipendente dalle CA;
  - gli utenti si fidano del server (ad es. server aziendale, trusted third party);
- **Delegated Responder**: il server OCSP è pagato da una CA:
  - risponde alle richieste solo sui certificati emessi dalla CA;
  - il certificato del server OCSP è fornito dalla CA (estensione "extended key usage" = `ocspSigning`);
  - gli utenti recuperano l'URL del server OCSP dall'estensione "authority information access" del certificato da verificare.

Attacchi DoS La firma della risposta causa un grosso carico sul server stesso ⇒ le risposte possono essere pre-calcolate per ridurre il carico sul server, ma:

- non sono più aggiornate in tempo reale;
- rendono possibili attacchi di tipo replay.

3.5 Time-stamping



> **time stamp** rispetto a un oggetto di dato, un'etichetta o un marchio in cui è registrato il tempo (ora del giorno o altro istante di tempo trascorso) in cui l'etichetta o il marchio è stato affisso all'oggetto di dato

Il **Time-Stamp Token** (TST) è una prova allegata a un documento per dimostrare che:

- il documento è stato creato prima di un certo istante di tempo (non è noto il tempo di creazione esatto);
- il documento non è stato modificato dopo quell'istante di tempo.

Il TST è fornito da un trusted third party, chiamato **Time-Stamping Authority** (TSA), tramite un protocollo client-server, chiamato **Time-Stamp Protocol** (TSP):

1. il client invia alla TSA il digest del documento calcolato tramite un algoritmo di hash (il documento potrebbe essere riservato);
2. la TSA restituisce un TST contenente:
   - il digest del documento;
   - la data e ora corrente fornita dall'orologio della TSA;
   - la firma digitale della TSA, calcolata sulla data e ora e sul digest;
3. il client allega il TST al documento.

Questo sistema è utile a livello applicativo quando un documento deve essere completato entro una certa scadenza, ma la consegna potrebbe essere ritardata (ad es. il server su cui caricarlo è sovraccarico).

Un singolo TST non è sufficiente per distinguere il momento della creazione e il momento della firma di un documento ⇒ con due TST si può dimostrare che il documento è stato firmato nell'intervallo di tempo compreso tra il primo e il secondo TST:

1. al documento privo di firma viene allegato un primo TST;
2. il documento viene firmato dall'utente con la propria chiave privata;
3. al documento firmato viene allegato un secondo TST che copre anche la firma.

3.6 PSE

personal security environment (PSE) archiviazione locale sicura per la chiave privata di un'entità, la chiave della CA direttamente fidata ed eventualmente altri dati; a seconda della politica di sicurezza dell'entità o dei requisiti di sistema, questo può essere, per esempio, un file protetto crittograficamente o un token hardware resistente alle manomissioni

Il **Personal Security Environment** è un luogo dove un utente può conservare:

- la propria chiave privata: deve essere tenuta segreta;
- i certificati self-signed delle CA radice fidate: devono essere autentici (non falsi).

Il PSE può essere implementato in due modi:

- **software** (sez. 3.8): la chiave privata e i certificati sono salvati all'interno di un file criptato;
- **hardware** (sez. 3.7): la chiave privata e i certificati sono salvati all'interno di un dispositivo hardware.

Un PSE supporta la **mobilità**: l'utente può usare la propria chiave su diversi dispositivi, anche se possono insorgere dei problemi:

- hardware: un dispositivo (ad es. smartphone) potrebbe non disporre di una porta USB a cui collegare il lettore di smart card;
- software: l'applicazione potrebbe non supportare il formato PKCS #12.

3.6.1 PKCS #11

PKCS #11 è una API di sicurezza di basso livello che offre agli sviluppatori delle applicazioni un'interfaccia standard per accedere alle primitive del PSE di livello inferiore.

Fornisce un **motore crittografico** in grado di svolgere le operazioni di crittografia base (ad es. firma, criptazione), che è implementabile:

- in software: una libreria che implementa tutti gli algoritmi;
- in hardware: una libreria che è uno stub verso un PSE hardware.

Crypto Service Provider (MS-CAPI CSP) è l'alternativa proprietaria di Microsoft.

3.6.2 Formati binari sicuri per i dati

Una migliore alternativa all'uso delle primitive crittografiche è rappresentata dai **formati binari** sicuri per i dati:

- PKCS #12 (sez. 3.8): formato che implementa i software PSE per la memorizzazione e il trasporto di chiavi private e certificati;
- PKCS #10 (sez. 3.9): formato per richieste di certificati in un'infrastruttura di certificazione;
- PKCS #7 (sez. 3.10): formato per buste sicure = struttura dati in cui mettere i dati da firmare e/o criptare.

Su questi formati binari si basano i **formati applicativi**, come lo standard S/MIME[4] e gli standard per la creazione di documenti elettronici a norma di legge.

[4]Si rimanda alla sezione 9.7.5.

3.7 PSE hardware

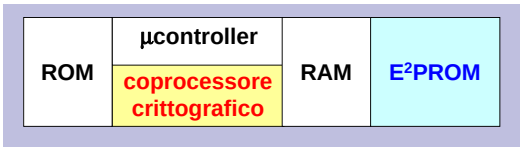
Un PSE hardware può essere:

- **passivo** (ad es. chiavetta USB): si limita a memorizzare la chiave privata e i certificati in forma criptata, ad es. firma digitale:
  1. il dispositivo fornisce la chiave privata in input al PC;
- **attivo**: è anche in grado di svolgere internamente le operazioni crittografiche grazie a funzioni crittografiche integrate a bordo, ad es. firma digitale:
  1. il PC fornisce il digest in input al dispositivo;
  2. il dispositivo restituisce la firma digitale in output al PC.

Le chiavi segrete possono essere generate:

- esternamente: la chiave deve essere generata esternamente e poi inserita nel dispositivo ⇒ in caso di guasto o smarrimento del dispositivo, i dati criptati sono ancora decriptabili se è stato effettuato il backup della chiave in un luogo sicuro;
- internamente: il dispositivo è in grado di calcolare la chiave autonomamente ⇒ la chiave rimane sempre confinata in un ambiente protetto (il PC potrebbe essere infettato da virus).

3.7.1 Smart card crittografica



token crittografico un dispositivo fisico, controllato dall'utente e portatile (ad es. smart card o carta PCMCIA) usato per memorizzare le informazioni crittografiche ed eventualmente anche svolgere le funzioni crittografiche

smart card crittografica carta a chip con memoria e con capacità crittografiche autonome

Una smart card crittografica contiene:

- microcontrollore: un semplice processore, con sistemi di I/O integrati, che avendo una potenza di calcolo limitata non è adatta a svolgere le operazioni crittografiche;
- ROM: memorizza il sistema operativo della smart card (CardOS);
- **coprocessore crittografico**: circuiti hardware che implementano le funzioni crittografiche (ad es. firma);
- **EEPROM**: una sorta di memoria flash non volatile, di piccole dimensioni (4-32 KB), in cui sono memorizzate le chiavi segrete in forma criptata.

Esistono vari tipi di smart card:

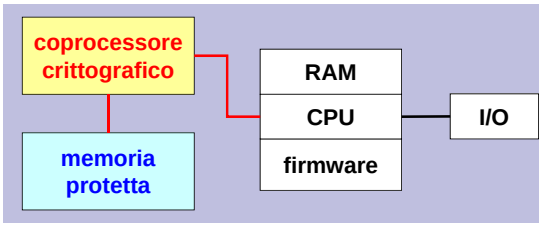
- semplici (economiche): il coprocessore crittografico implementa un algoritmo simmetrico (DES);
- complesse (costose): il coprocessore crittografico implementa un algoritmo asimmetrico (RSA) o a curve ellittiche.
  Il costo dipende da:
  - lunghezza delle chiavi;
  - generazione delle chiavi private a bordo.

Svantaggi

- l'utilizzo delle smart card richiede un **lettore di smart card**:
  - sono necessari driver per tutti i sistemi operativi esistenti;
  - è necessario il supporto da parte delle applicazioni.
- le smart card sono per gli utenti, non per i server:
  - le interfacce di I/O consentono dei trasferimenti lenti;
  - le operazioni crittografiche sono lente.

  Un attaccante potrebbe inviare a una smart card attiva tanti digest da firmare per rendere la smart card occupata ⇒ si può chiedere all'utente di digitare un **pin**, per mezzo di un tastierino a bordo, ogni volta che viene generata una firma.

3.7.2 HSM

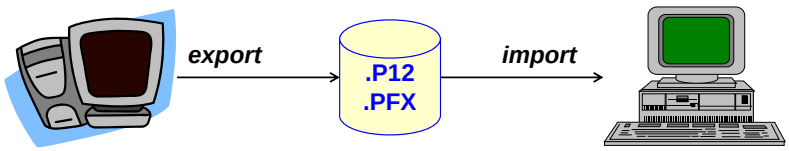


L'**Hardware Security Module** (HSM) è un acceleratore crittografico per server (ad es. server OCSP, Time-Stamping Authority) che ha gli stessi componenti logici della smart card, ma con prestazioni migliori.

Fattori di forma

- schede PCI;
- dispositivi esterni: USB, IP, SCSI...

3.8 PKCS #12



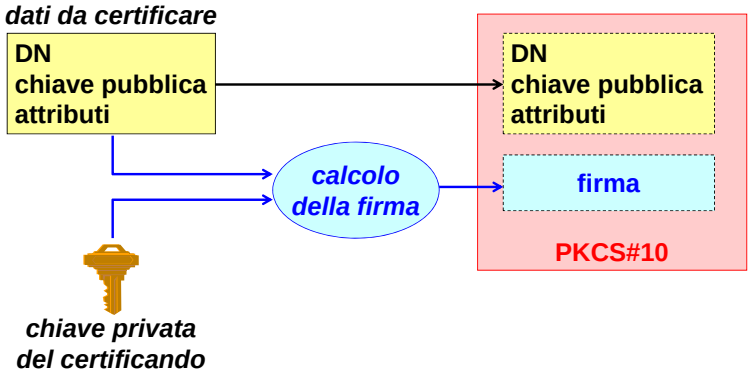
PKCS #12 è il formato standard, derivato dal formato PFX di Microsoft, che implementa un PSE software:

- trasporto: l'identità digitale può essere esportata e importata tra applicazioni (ad es. browser Web) o sistemi diversi;
- backup: la chiave privata può essere recuperata in caso di smarrimento.

Un file PKCS #12 contiene del materiale crittografico personale, firmato e criptato tramite password, come:

- la chiave privata;
- il certificato corrispondente alla chiave privata;
- il certificato della CA emittente.

3.9 PKCS #10



PKCS #10 è il formato standard usato da un utente per la richiesta di un certificato alla CA.
La richiesta PKCS #10 contiene:

- il distinguished name (DN) dell'utente richiedente;
- la chiave pubblica da certificare;
- eventuali attributi, tra cui:
  - una password a sfida tramite la quale l'utente potrà in seguito richiedere la revoca del certificato;
  - gli attributi (ad es. PKCS #9) che dovranno essere inseriti nel certificato;
  - altre informazioni sul richiedente.

protocollo proof-of-possession un protocollo con il quale un'entità di sistema dimostra a un'altra di possedere e controllare una chiave crittografica o altre informazioni segrete

La richiesta PKCS #10 viene firmata con la chiave privata corrispondente alla chiave pubblica da certificare ⇒ la CA può verificare che il richiedente sia il legittimo titolare della chiave pubblica.

3.10 PKCS #7

PKCS #7, denominato "Cryptographic Message Syntax" (CMS), permette di creare **buste sicure** che possono contenere dati di qualsiasi tipo.
PKCS #7 è molto flessibile:

- i dati possono essere criptati e/o firmati usando algoritmi simmetrici o asimmetrici;
- permette più firme (parallele o sequenziali) su uno stesso oggetto (si rimanda alla sezione 3.11.2);
- è un **formato autoconsistente**: può includere, oltre ai dati e alla firma, anche i certificati (con le relative informazioni di revoca) necessari per verificare la firma;
- è un **formato ricorsivo**: il contenuto di un oggetto PKCS #7 può essere un altro oggetto PKCS #7.

Una busta PKCS #7 contiene un blocco di alto livello (`contentInfo`) suddiviso in due parti:

- `contentType`: il tipo di contenuto base;

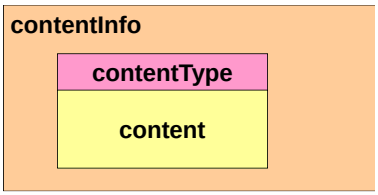


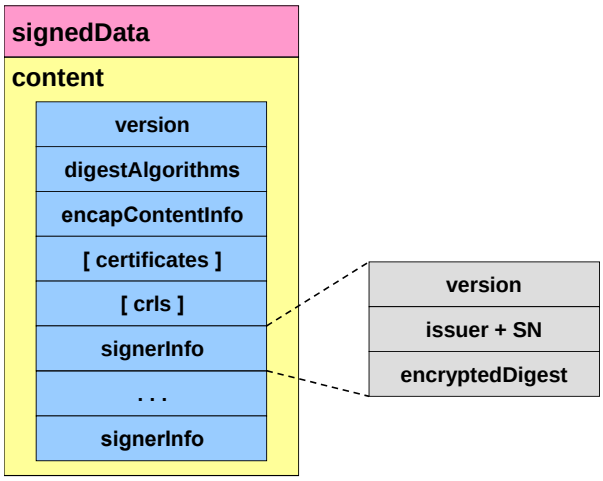
Figura 3.4: Struttura PKCS #7.

- `content`: i dati + le informazioni aggiuntive sui dati.

Lo standard PKCS #7 definisce sei tipi di contenuto base:

- `data`: dati in chiaro (codifica di una generica sequenza di byte);
- `signedData` (sez. 3.10.1): dati in chiaro + una o più firme parallele;
- `envelopedData` (sez. 3.10.2): dati criptati con una chiave simmetrica + chiave simmetrica criptata con le chiavi pubbliche RSA dei destinatari;
- `signedAndEnvelopedData`: dati + firme digitali, entrambi criptati con le chiavi pubbliche RSA dei destinatari;
- `digestData`: dati in chiaro + digest (solo per integrità);
- `encryptedData`: dati criptati con una chiave simmetrica (non inclusa perché assunta gestita per altri mezzi).

3.10.1 Documenti firmati digitalmente (enveloping signature)



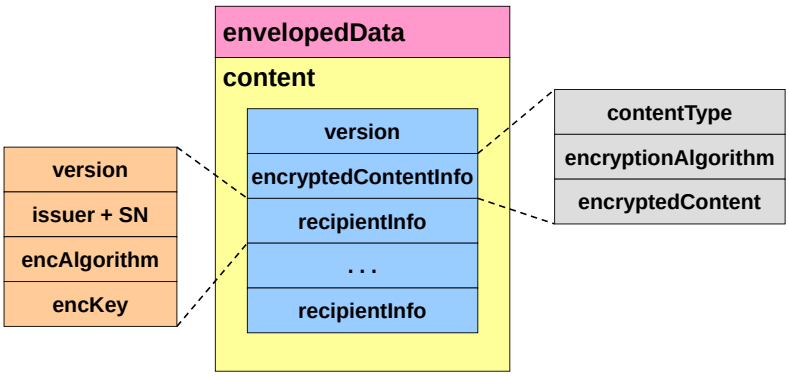
Il tipo `signedData` è il più usato per imbustare dei dati firmati:

- `version`: la versione dello standard usato;
- `digestAlgorithms`: gli algoritmi di hash usati per creare le firme;
- `encapContentInfo`: la struttura contenente i dati incapsulati (in chiaro);
- `certificates` (facoltativo): l'elenco dei certificati necessari per verificare le firme;
- `crls` (facoltativo): l'elenco delle CRL necessarie per verificare le firme;

- `signerInfo`: le informazioni su uno dei firmatari:
  - `version`: il numero di versione;
  - `issuer + SN`: il nome dell'emettitore e il numero seriale di certificato che identificano univocamente il certificato del firmatario;
  - `encryptedDigest`: la firma digitale.

Il campo `digestAlgorithms` è posto all'inizio della busta ⇒ il ricevente, durante la lettura dei dati incapsulati, può subito iniziare a calcolare i digest su di essi, per poterli poi confrontare velocemente con quelli contenuti nella busta.

3.10.2 Buste digitali



busta digitale una combinazione di

(a) dati (di qualsiasi tipo) con contenuto criptato intesi per un destinatario

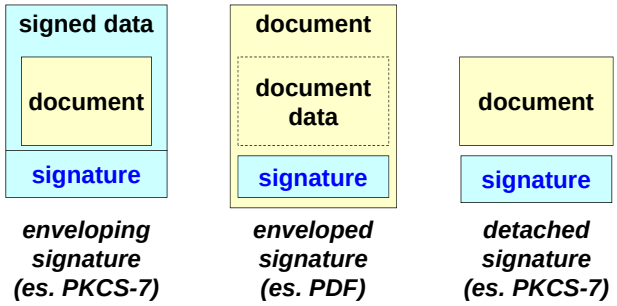
(b) la chiave di criptazione del contenuto in forma criptata che è stata preparata per l'uso da parte del destinatario

Il tipo `envelopedData` è il più usato per imbustare dei dati criptati:

- `version`: la versione dello standard usato;
- `encryptedContentInfo`: la struttura contenente i dati incapsulati criptati:
  - `contentType`: il tipo di contenuto;
  - `encryptionAlgorithm`: l'algoritmo di crittografia simmetrica usato per criptare i dati;
  - `encryptedContent`: i dati incapsulati veri e propri, criptati con la chiave simmetrica;
- `recipientInfo`: le informazioni su uno dei destinatari (= riceventi autorizzati a leggere il contenuto):
  - `version`: il numero di versione;
  - `issuer + SN`: il nome dell'emettitore e il numero seriale di certificato che identificano univocamente il certificato del destinatario contenente la chiave pubblica usata per criptare la chiave simmetrica;
  - `encAlgorithm`: l'algoritmo di crittografia asimmetrica usato per criptare la chiave simmetrica;
  - `encKey`: la chiave simmetrica usata per criptare i dati, criptata con la chiave pubblica del destinatario.

3.11 Documenti firmati digitalmente

3.11.1 Formati di documenti firmati

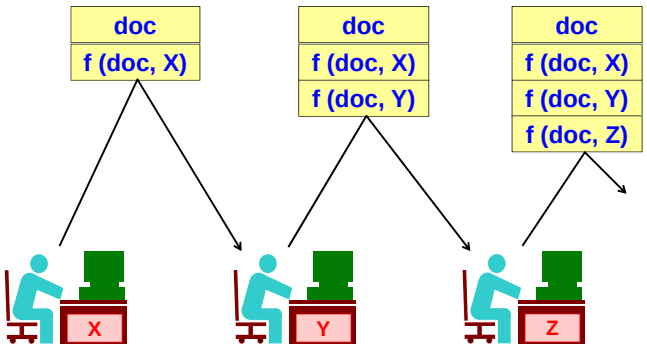


A livello concettuale, la firma digitale può essere allegata al documento in tre formati:

- **enveloping signature** (ad es. PKCS #7: sez. 3.10.1): la firma racchiude il documento ⇒ il documento va estratto prima di poter essere letto;
- **enveloped signature** (ad es. PDF): il documento racchiude la firma, cioè il formato del documento prevede uno spazio per la firma;
- **detached signature** (ad es. PKCS #7): il documento e la firma sono separati tra loro, e la firma contiene un riferimento (ad es. nome del file) al documento ⇒ problema: come mantenere nel tempo l'associazione tra la firma e il documento?

3.11.2 Firme multiple

Firme parallele

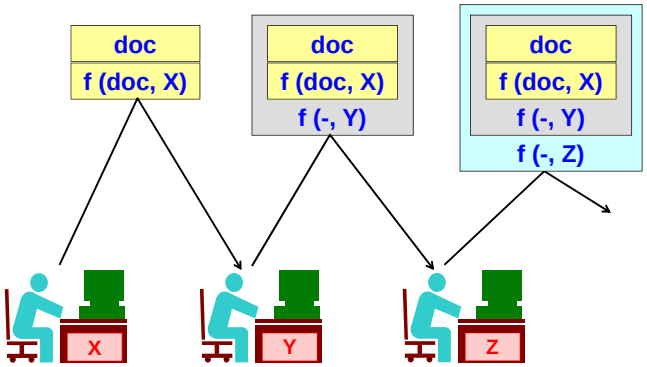


Una singola busta PKCS #7 può contenere più **firme parallele** (o indipendenti):

1. l'utente X crea il documento e lo firma;
2. l'utente Y aggiunge la propria firma relativa al documento;
3. l'utente Z aggiunge la propria firma relativa al documento, e così via.

Svantaggio Non è possibile sapere in quale ordine sono state apposte le firme.

Firme sequenziali



Più buste PKCS #7 possono essere contenute ricorsivamente l'una dentro l'altra, ognuna contenente una delle **firme sequenziali** (o gerarchiche):

1. l'utente X crea il documento e lo firma;
2. l'utente Y aggiunge la propria firma relativa al documento compresa la firma dell'utente X;
3. l'utente Z aggiunge la propria firma relativa al documento comprese le firme precedenti, e così via.

Vantaggio È possibile sapere in quale ordine sono state apposte le firme, in quanto i firmatari sono legati tra loro da una gerarchia (ad es. ricercatore → capo del dipartimento → rettore dell'università).

3.11.3 Valore legale

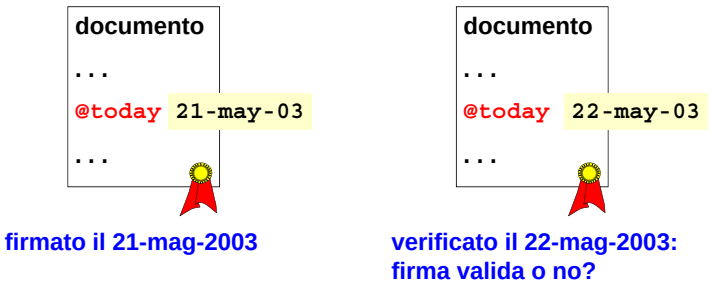


Figura 3.5: La firma è valida, nonostante la presenza della macro: non sono cambiati i bit, ma solo la visualizzazione finale del documento.

Una firma digitale offre la proprietà di non ripudio, ma molti aspetti devono essere considerati da una corte di giustizia affinché ne venga riconosciuto **valore legale**:

- sintassi: è la tua firma? (tutti i bit devono corrispondere)
- semantica: hai capito ciò che stavi firmando? (ad es. il testo bianco su sfondo bianco non rispetta il WYSIWYS = "What You See Is What You Sign")
- volontà: hai firmato volontariamente? (un notaio deve verificare la volontà del contraente)
- identificazione: sei stato tu a firmare? (ad es. smart card rubata; un notaio deve identificare il contraente)

- tempo: quando hai firmato? (la data può essere manipolata)
- luogo: dove hai firmato? (se il documento elettronico ha una qualche valenza fiscale, Paesi diversi possono avere tassazioni diverse)

3.12 Firma elettronica europea

> **firma elettronica** l'insieme dei dati in formato elettronico che sono attaccati o logicamente associati con altri dati in formato elettronico e che ne forniscono un mezzo di autenticazione

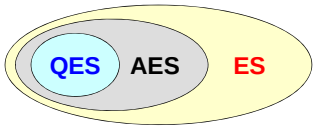
La **firma elettronica** (ES) è una firma che ha valore legale a livello di Unione Europea.

La ES è un concetto generale che può comprendere vari tipi di firme con diversi livelli di incertezza:

- una firma digitale;
- una firma autografa scandita;
- il codice restituito da una penna speciale che riconosce i movimenti della mano.

Gli Stati membri devono assicurare che ad una ES non sia negato valore legale solo perché:

- è in forma elettronica;
- non è basata su Qualified Certificate;
- non usa certificati emessi da certificatori autorizzati;
- non è stata creata con un dispositivo di firma sicuro.



A seconda del livello di incertezza:

- AES (sez. 3.12.1): il suo valore legale deve essere discusso da una corte di giustizia;
- QES (sez. 3.12.2: ha automaticamente valore legale equivalente alla firma autografa, grazie al livello di incertezza minimo.

3.12.1 AES

Una **Advanced Electronic Signature** (AES) è una ES che soddisfa i seguenti requisiti:

- è in relazione univoca con il firmatario (solo lui può averla fatta);
- consente di identificare il firmatario;
- è stata creata usando strumenti che il firmatario può mantenere sotto il suo controllo (ad es. smart card);
- è in relazione con i dati ai quali si riferisce in modo che ogni successiva modifica dei dati possa essere individuata (ad es. digest).

Questi requisiti non sono legati a una certa tecnologia, ma sono stati definiti in maniera indipendente dall'evoluzione tecnologica per continuare a essere validi in futuro.

3.12.2 QES

Una **Qualified Electronic Signature** (QES) è una AES che soddisfa i seguenti requisiti:

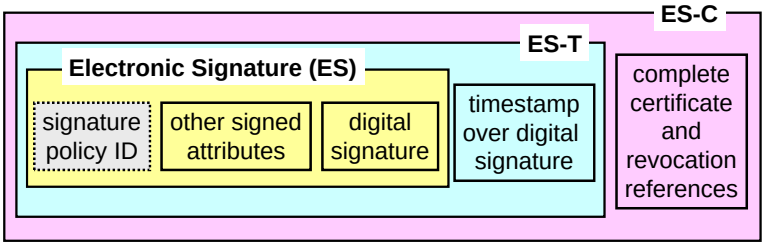
- è stata apposta usando un Qualified Certificate;
- è stata apposta usando un dispositivo di firma certificato essere sicuro.

> **qualified certificate** un certificato a chiave pubblica che ha lo scopo primario di identificare una persona con un livello di certezza elevato, dove il certificato soddisfa alcuni requisiti di qualificazione definiti da un quadro giuridico applicabile, come la Direttiva Europea sulla Firma Elettronica

Un **Qualified Certificate** (QC) è un certificato che garantisce l'identità di una persona e contiene:

- l'indicazione che si tratta di un QC;
- l'indicazione del certificatore e dello stato in cui è stato emesso;
- indicazioni sulle limitazioni di utilizzo del certificato;
- indicazioni sul limite delle transazioni commerciali effettuabili con quel certificato.

3.12.3 CAdES



Il **CMS Advanced Electronic Signatures** (CAdES) è il formato standard europeo per le firme elettroniche, pubblicato dall'ente ETSI e basato sul formato CMS (PKCS #7).

Tipi di formati CAdES

- ES: prevede:
  - firma digitale;
  - altri attributi firmati;
  - identificativo della politica di firma;
- ES-T (timestamp): sulla firma è calcolato un timestamp per impedire attacchi back-dating;
- ES-C (complete): contiene, oltre alla firma con timestamp, anche i riferimenti a tutti i certificati e le informazioni di revoca necessari per verificare la validità della firma;
- ES-X (extended): offrono un'ulteriore sicurezza quando i certificati delle CA possono essere compromessi:
  - ES-X-Timestamp (type 1) (con OCSP): il timestamp è applicato all'intera ES-C (OCSP);
  - ES-X-Timestamp (type 2) (con CRL): il timestamp è applicato solo ai riferimenti ai certificati e alle informazioni di revoca.

Formati di firma applicativi CAdES è un formato di firma binario da cui derivano alcuni formati di firma applicativi, come:

- XML Advanced Electronic Signatures (XAdES): basato su XML-dsig, standard di firma digitale per documenti XML;
- PDF Advanced Electronic Signature Profiles (PAdES): standard per le firme elettroniche nei documenti PDF.

Capitolo 4

Sistemi di autenticazione

4.1 Metodologie di autenticazione

Le metodologie di autenticazione possono essere basate su meccanismi diversi:

1. "qualcosa che l'utente conosce" (ad es. password, PIN);
2. "qualcosa di cui l'utente è in possesso" (ad es. carta magnetica);
3. "qualcosa che l'utente è fisicamente" (biometria, ad es. impronta digitale).

Meccanismi diversi possono essere combinati per avere un livello autenticazione sempre crescente, fino alla reale identificazione[1]:

- **autenticazione a un fattore**: primo meccanismo;
- **autenticazione a due fattori**: primo meccanismo + secondo meccanismo (ad es. bancomat);
- **autenticazione a tre fattori**: primo meccanismo + secondo meccanismo + terzo meccanismo.

Con la diffusione dei cellulari si potrebbe aggiungere un altro meccanismo, cioè la posizione dell'utente, sempre nota dalla rete cellulare.

4.1.1 Autenticazione a un fattore

autenticazione il processo di verifica di una dichiarazione che un'entità di sistema o una risorsa di sistema ha un certo valore di attributo

identificazione un atto o processo che presenta un identificativo a un sistema in modo che il sistema possa riconoscere un'entità di sistema e distinguerla dalle altre entità

verifica il processo di esaminamento delle informazioni per stabilire la verità di un dato o valore dichiarato

Il sistema è costituito da:

- utente: è identificato da uno User ID (UID), ed è a conoscenza di un segreto S_{UID} (ad es. password);

[1] L'autenticazione è un concetto diverso dall'identificazione:

- autenticazione: è il risultato di una procedura elettronica (ad es. inserimento di nome utente e password);
- identificazione: significa avere la certezza che le credenziali di accesso siano state effettivamente inserite dal loro titolare e non da persone terze.

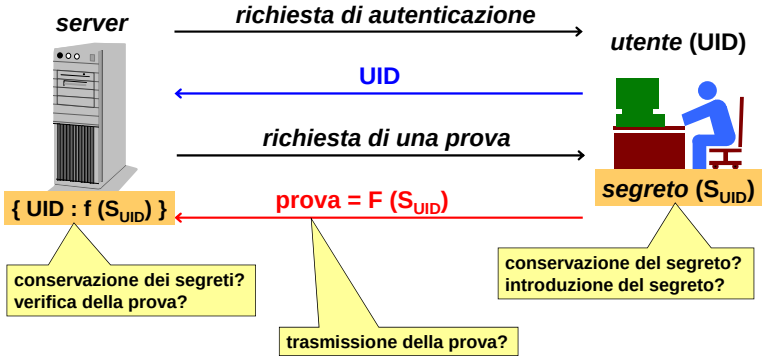


Figura 4.1: Schema di riferimento per un sistema di autenticazione basato sul primo meccanismo.

- server: contiene una tabella con delle coppie $\{\text{UID} : f(S_{\text{UID}})\}$, dove f è una funzione calcolata sul segreto.

Il processo di autenticazione agisce prima dell'inizio della comunicazione:

1. **identificazione**: presentazione del valore di attributo dichiarato al sottosistema di autenticazione:
   (a) richiesta di autenticazione: il server chiede all'utente di identificarsi;
   (b) UID: l'utente comunica al server il proprio UID (ad es. nome utente);
2. **verifica**: presentazione o generazione delle informazioni di autenticazione che agiscono come prova per dimostrare l'associazione tra l'attributo e ciò per cui è stato dichiarato:
   (a) richiesta di una prova: il server richiede una prova all'utente, che dimostri che lui sia davvero chi sta dichiarando di essere;
   (b) prova: l'utente fornisce come prova il risultato della funzione F applicata al segreto S_{UID}: prova $= F(S_{\text{UID}})$;
   (c) controllo: il server confronta la prova ricevuta con il segreto S_{UID} memorizzato: se la prova è quella attesa, concede all'utente l'accesso ai servizi che offre.

Problemi

- lato client:
  - l'utente come inserisce il segreto da inviare al server?
  - il segreto dell'utente come può essere tenuto al sicuro?
- lato rete: la prova potrebbe venire cancellata o manipolata a piacere da un man-in-the-middle: come può essere trasmessa in modo sicuro?
- lato server:
  - il server come può verificare la prova?
  - ha bisogno di conoscere il segreto dell'utente: questo segreto come può essere memorizzato nel server in modo sicuro?

4.2 Memorizzazione delle password sul server

Come salvare le password sul server in modo sicuro?

- in chiaro: tutte le password sono memorizzate in chiaro (f = funzione identità I):

$$f(S_{\text{UID}}) = P_{\text{UID}}$$

  - − chiunque ha accesso al server è in grado di leggere le password di tutti gli utenti;

- criptate: tutte le password sono memorizzate criptate (f = funzione di criptazione enc):

$$f(S_{\text{UID}}) = \text{enc}(K, P_{\text{UID}})$$

  - − la chiave di decriptazione deve essere salvata in chiaro;

- con hash: sono memorizzati solo i digest delle password (f = funzione di hash h):

$$f(S_{\text{UID}}) = h(P_{\text{UID}})$$

  - + il digest è difficilmente invertibile ⇒ è impossibile risalire alla password.
  - − i digest non protetti sono soggetti ad attacchi a dizionario;

- con hash con sale: sono memorizzati dei digest imprevedibili (f = funzione di hash h):

$$f(S_{\text{UID}}) = h(P_{\text{UID}} || \text{sale})$$

  - + gli attacchi a dizionario, anche con rainbow table, sono resi praticamente impossibili.

4.2.1 Attacco a dizionario

attacco a dizionario un attacco che usa una tecnica a forza bruta provando in successione tutte le parole in una grande lista esaustiva

Un **attacco a dizionario** è un tipo di attacco a forza bruta con lo scopo di scoprire una password conoscendone l'hash (= digest): viene usata una lista, detta **dizionario**, che contiene solo le password più probabili, ovvero quelle che gli esseri umani scelgono più di frequente perché facili da ricordare.

Ipotesi L'attaccante ha accesso alle informazioni nel server:

- ha il database del server contenente gli hash corrispondenti alle password degli utenti;
- conosce l'algoritmo di hash usato per calcolare gli hash.

Precalcolo Per ogni parola p contenuta all'interno di un dizionario, l'attaccante:

```
for (each p in Dictionary)
   store(DB, p, h(p))
```

1. calcola l'hash h della parola;
2. memorizza la coppia $\{p, h\}$ in un database.

Attacco Per ogni hash HP nel database del server, l'attaccante effettua una ricerca esaustiva nel database precalcolato:

```
for (each HP in ServerDB)
   for (each (p, h) in DB) // ricerca nel database
      if (h == HP) // password trovata
         write("password = " + p)
         go to next HP
```

Casi di studio

- MySQL: a partire dalla versione 4.1, MySQL usa un **doppio hash** con SHA1 (senza sale) ⇒ il doppio hash non è sufficiente a impedire gli attacchi a dizionario: basta creare il database precalcolato applicando due volte la funzione di hash;
- attacco a LinkedIn (giugno 2012): siccome i cybercriminali non avevano a disposizione i dizionari pronti, sono ricorsi al **crowdsourcing**: hanno pubblicato l'elenco di tutti gli hash su un sito Web, e hanno chiesto agli utenti del sito di contribuire al cracking delle password eseguendo sui propri PC un programma allegato all'elenco.

4.2.2 Rainbow table

Le **rainbow table** sono degli speciali database precalcolati che permettono di raggiungere un compromesso tra:

- memoria: solo un sottoinsieme degli hash è salvato nel database precalcolato;
- tempo di calcolo: quando la catena viene trovata, occorre ripercorrere la catena per trovare la password.

Ipotesi

- l'attaccante ha accesso alle informazioni nel server (come nell'attacco a dizionario);
- per semplicità, tutte le password sono composte da 5 cifre ⇒ esistono 10^5 password possibili;
- l'attaccante sceglie di costruire tabelle di rainbow composte da 100 righe ⇒ ogni riga rappresenta una catena di 1000 password;
- l'attaccante sceglie una funzione di riduzione r che, dato un hash h, restituisce una password p qualunque (che non necessariamente corrisponde all'hash h).

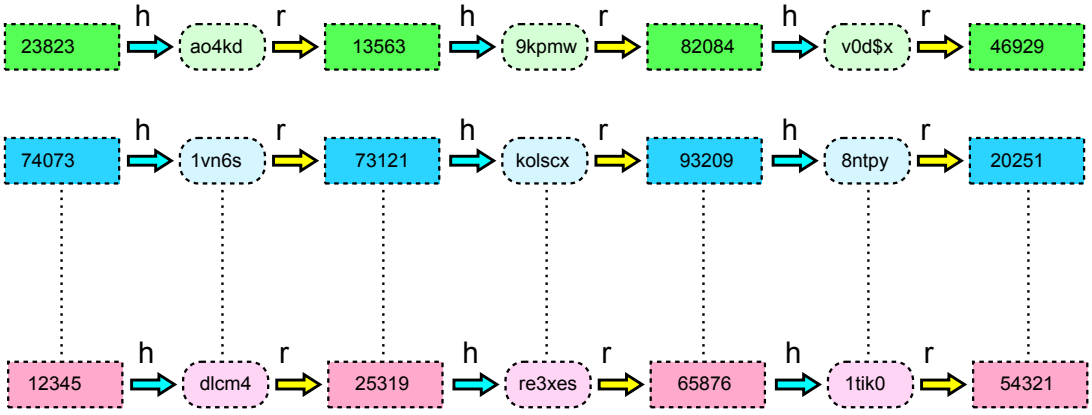


Figura 4.2: Esempio di rainbow table.[2]

[2]Questa immagine è derivata da un'immagine su Wikimedia Commons (Rainbow table1.svg), realizzata dall'utente Dake, ed è concessa sotto la licenza Creative Commons Attribuzione - Condividi allo stesso modo 4.0 Internazionale.

Precalcolo L'attaccante:

```
for (100 distinct P)
    p = P
    for (i = 0 : 1000)
        k = h(p)
        p = r(k)
    store(RT, P, p)
```

1. prende 100 password P distinte qualsiasi tra le 10^5 possibili;
2. a partire da ognuna di queste password, costruisce una catena che passa attraverso 1000 password alternando le funzioni di hash h e di riduzione r;
3. memorizza l'ultima password p (coda) ottenuta alla fine di ogni catena, insieme alla password P di partenza (testa), nella rainbow table, che alla fine conterrà 10^2 coppie $\{P, p\}$.

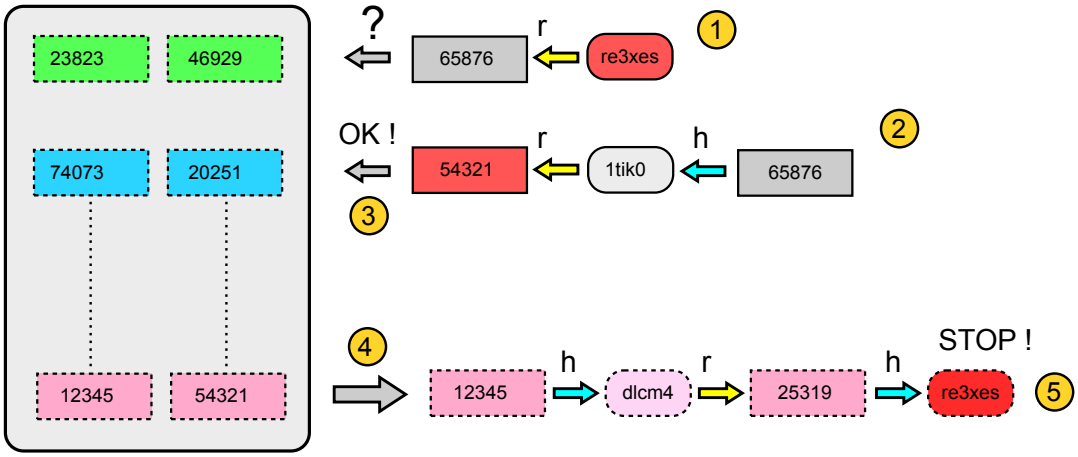


Figura 4.3: Esempio di attacco a dizionario con rainbow table (hash = `re3xes`*, password =* `25319`*).*[3]

Attacco Per ogni hash HP nel database del server, l'attaccante:

```
for (each HP in ServerDB)
    for (i = 0 : 1000)
        h = (i == 0) ? HP : h(ph)
        ph = r(h)
        for (each (P, p) in RT) // ricerca nella rainbow table
            if (p == ph) // catena trovata
                for (i = 0 : 1000) // ricerca nella catena
                    Ph = (i == 0) ? P : r(h)
                    h = h(Ph);
                    if (h == HP) // password trovata
                        write("password = " + Ph)
                        go to next HP
```

[3]Questa immagine è derivata da un'immagine su Wikimedia Commons (Rainbow table2.svg), realizzata dall'utente Dake, ed è concessa sotto la licenza Creative Commons Attribuzione - Condividi allo stesso modo 4.0 Internazionale.

1. a partire dall'hash HP, costruisce una catena che passa attraverso al più 1000 password alternando le funzioni di riduzione r e di hash h;
2. confronta ogni password ph intermedia con le code p di tutte le catene nella rainbow table;
3. una volta trovata la catena, ripercorre la catena dalla testa P alla coda p in modo analogo al precalcolo;
4. confronta ogni hash h intermedio con l'hash HP;
5. una volta trovato l'hash HP nella catena, l'ultima password Ph su cui è stata applicata la funzione di hash è la password dell'utente.

Miglioramenti

- per evitare che la stessa password appaia in più catene, si possono usare funzioni di riduzione diverse $r_0() \dots r_n()$;
- esistono in vendita rainbow table precalcolate per varie funzioni di hash e vari insiemi di password.

4.2.3 Scelta della password

Gli utenti tendono a scegliere password facilmente memorizzabili, e quindi semplici da indovinare. Le password dovrebbero non essere usate affatto, ma l'uso di almeno una password è inevitabile, a meno di non ricorrere a sistemi biometrici.

Come scegliere una password robusta?

- dovrebbe usare tipi diversi di caratteri (alfabetici maiuscoli e minuscoli + cifre + caratteri speciali);
- dovrebbe essere più lunga possibile (almeno 8 caratteri);
- non dovrebbe essere presente in dizionario;
- dovrebbe essere cambiata periodicamente.

ransomware un tipo di malware che limita l'accesso al sistema di elaborazione che infetta, e chiede un riscatto pagato al creatore/i del malware affinché la restrizione venga rimossa

Caso di studio attacco a iCloud (maggio 2014): alcuni cybercriminali hanno violato alcuni account iCloud protetti da password deboli, quindi hanno usato il servizio "Find My Device" per mettere da remoto i dispositivi Apple degli utenti in "Lost Mode", chiedendo il pagamento di un riscatto per sbloccarli.

4.2.4 Sale

sale un valore di dato usato per variare i risultati di un calcolo in un meccanismo di sicurezza, in modo che un risultato computazionale esposto di una istanza di applicazione del meccanismo non possa essere riutilizzato da un attaccante in un'altra istanza

Il **sale** è un numero usato per variare la password prima di calcolare l'hash.

Hashing con sale

1. viene scelto come sale un numero:
   - casuale: varia la password in modo imprevedibile (meglio se con caratteri poco usati o di controllo);
   - lungo: aumenta la complessità del dizionario compensando l'uso di password brevi;
   - nonce: è diverso per ciascun UID ⇒ se due utenti usano la stessa password, i loro hash saranno diversi;
2. l'hash HP è calcolato sulla password concatenata al sale:
$$\text{HP} = \text{hash}\ (\ \text{password} \,||\, \text{sale}\ )$$
3. sul server è memorizzato anche il sale in chiaro, per poter successivamente verificare la password:
$$\{\ \text{UID},\ \text{HP}_{\text{UID}},\ \text{sale}_{\text{UID}}\ \}$$

Vantaggi

- se due utenti usano la stessa password, i loro hash saranno diversi ⇒ uno non può scoprire di avere la stessa password dell'altro;
- gli attacchi a dizionario, anche con rainbow table, sono resi praticamente impossibili:
  - sarebbe necessario un database precalcolato per ogni possibile sale;
  - se l'utente cambia periodicamente la password (e quindi il sale), l'attaccante non ha il tempo per creare il database precalcolato.

KDF

Una chiave crittografica deve essere completamente casuale, cioè ogni bit deve avere il 50% di probabilità di essere 0 o 1, per essere imprevedibile e difficile da indovinare. Tuttavia gli esseri umani non sono in generale molto bravi a inventare dei dati casuali.

Gli algoritmi di hash, oltre a garantire l'integrità dei dati, possono essere usati anche per generare delle chiavi crittografiche casuali a partire da una password segreta ⇒ l'utente non deve avere a che fare con una sequenza di bit senza alcun significato, mentre una password, anche poco casuale, può essere memorizzata e digitata facilmente con la tastiera.

Una **Key Derivation Function** (KDF) è in grado di derivare automaticamente una o più chiavi segrete da una password segreta usando una funzione di hash:

$$K = \text{KDF}\,(P, S, I)$$

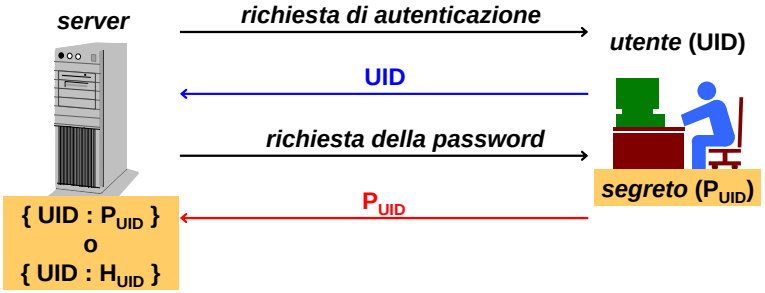
dove:

- K è l'insieme delle chiavi crittografiche generate;
- P è la password/passphrase fornita in input dall'utente;
- S è il sale, distinto in base all'utente;
- I è il numero di iterazioni della funzione base, al fine di rallentare il calcolo e rendere un possibile attacco più complicato.

Le KDF più usate sono basate su funzioni di hash crittografiche:

- PBKDF2: usa SHA-1 ($|S| \geq 64$ bit, $I \geq 1000$);
- HKDF: è basato su HMAC.

4.3 Autenticazione basata su password ripetibili



Ciò che l'utente conosce è una **password ripetibile**: l'utente può riutilizzarla quante volte vuole.

Procedura di autenticazione

1. identificazione: il client invia il proprio UID;
2. richiesta di una prova: il server chiede una password;
3. prova: il client trasmette la password in chiaro (F = funzione identità I):
$$F\left(S_{\text{UID}}\right) = P_{\text{UID}}$$
4. controllo: il server:
   (a) calcola l'hash con sale sulla prova ricevuta;
   (b) confronta l'hash calcolato con il digest memorizzato:
$$\text{hash}\left(\text{prova} \,||\, \text{sale}_{\text{UID}}\right) = \text{HP}_{\text{UID}}?$$

Svantaggi

- lato client: la responsabilità della conservazione della password è lasciata all'utente:
  - l'utente potrebbe scrivere la password in un posto non sicuro (ad es. post-it);
  - gli utenti tendono a scegliere password facilmente memorizzabili, e quindi semplici da indovinare;
- lato rete:
  - attacchi di sniffing: il segreto dell'utente è trasmesso in chiaro;
  - attacchi replay: la password è valida per più accessi;
- lato server: occorre usare il sale per proteggere le password memorizzate sul server $\Rightarrow$ la verifica richiede il calcolo del digest di ogni prova.

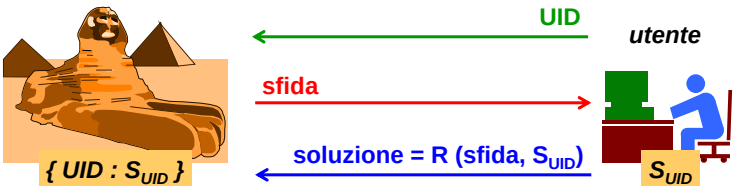
4.3.1 Sistemi a sfida

> **a sfida** un processo di autenticazione che verifica un'identità richiedendo che siano fornite le informazioni di autenticazione corrette in risposta a una sfida [...]

Per evitare che la prova venga intercettata durante la sua trasmissione, occorre ricorrere ai **sistemi a sfida**:

- simmetrici (sez. 4.4): l'utente risolve la sfida calcolandone il keyed-digest usando il segreto condiviso con il server;
- asimmetrici (sez. 4.5): l'utente risolve alla sfida decriptandola usando la propria chiave privata.

4.4 Sistemi a sfida simmetrici



Ciò che l'utente conosce è un segreto condiviso: l'utente dimostra di conoscere il segreto senza trasmetterlo, ma usandolo per effettuare un calcolo.

Procedura di autenticazione

1. identificazione: il client invia il proprio UID;
2. richiesta di una prova: il server:
   (a) genera un numero nonce e casuale;
   (b) invia il numero generato come **sfida**;
3. prova: il client:
   (a) calcola il keyed-digest sulla sfida ricevuta, usando il segreto condiviso come chiave (F = funzione di hash R):
   $$F\left(S_{\text{UID}}\right) = R\left(\text{sfida}, S_{\text{UID}}\right)$$
   (b) trasmette il keyed-digest calcolato come soluzione alla sfida;
4. controllo: il server:
   (a) esegue lo stesso calcolo della soluzione usando la sfida generata e il segreto condiviso;
   (b) confronta la prova ricevuta con la soluzione calcolata:
   $$\text{prova} = R\left(\text{sfida}, S_{\text{UID}}\right)?$$

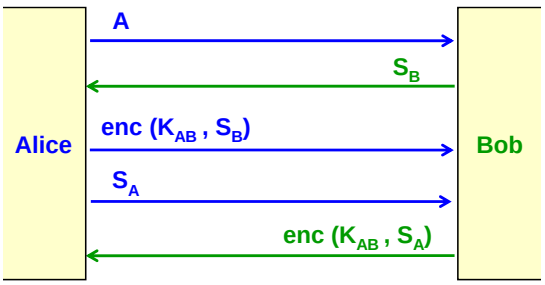
Vantaggi

- no attacchi di sniffing: non è possibile risalire al segreto dell'utente;
- no attacchi replay: la sfida contiene un numero nonce.

Svantaggi

- lato client: l'utente deve avere a disposizione una funzione di hash per eseguire il calcolo della soluzione ⇒ se sta usando un terminale pubblico e non possiede una funzione di hash, non potrà rispondere alla sfida;
- lato server: il server non può memorizzare gli hash delle password, ma deve conoscere in chiaro il segreto per poter eseguire lo stesso calcolo della soluzione.

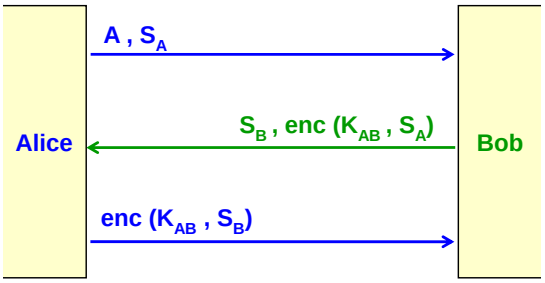
4.4.1 Mutua autenticazione con protocolli a sfida simmetrici



I sistemi a sfida simmetrici si prestano bene a fare **mutua autenticazione** tra due peer, dove anche il server (Bob) viene autenticato:

1. Alice inizia lo scambio inviando il proprio UID;
2. Bob invia ad Alice una sfida S_B;
3. Alice invia la soluzione alla sfida S_B calcolata usando la chiave comune K_{AB};
4. Alice invia a Bob una sfida S_A;
5. Bob invia la soluzione alla sfida S_A calcolata usando la chiave comune K_{AB}.

Ottimizzazione



Riducendo il numero di messaggi scambiati si possono migliorare le prestazioni (ma non la sicurezza):

1. Alice inizia lo scambio inviando il proprio UID, insieme alla sua sfida S_A;
2. Bob invia la soluzione alla sfida S_A, insieme alla sua sfida S_B;
3. Alice invia la soluzione alla sfida S_B.

4.4.2 Possibili errori di implementazione

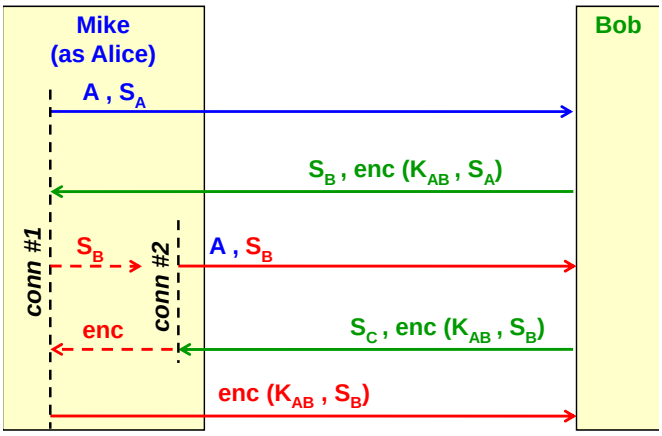
I protocolli a sfida simmetrici sono di semplice implementazione, ma se implementati scorrettamente possono essere soggetti ad attacchi:

- **funzione di hash invertibile**: un attaccante potrebbe risalire al segreto intercettando la sfida e la soluzione:
  - soluzione: la funzione R usata per calcolare la soluzione deve essere una funzione di hash crittografico;
- **sfida ripetuta**: un attaccante potrebbe ripetere la risposta (attacco replay):

  - soluzione: la sfida deve essere un nonce, in modo che le soluzioni siano diverse a ogni autenticazione di un utente;

- **sfida prevedibile** (ad es. basata su data e ora): un attaccante potrebbe apprendere la soluzione della prossima sfida facendo autenticare il client su un server fittizio:
  - soluzione: la sfida deve essere casuale;
- **doppia autenticazione**: un attaccante potrebbe apprendere la soluzione della sfida corrente facendo autenticare il server due volte:
  - soluzione 1: il server deve consentire l'apertura di una connessione per volta;
  - soluzione 2: la chiave comune può essere sostituita con due chiavi distinte unidirezionali.

Attacco con doppia autenticazione

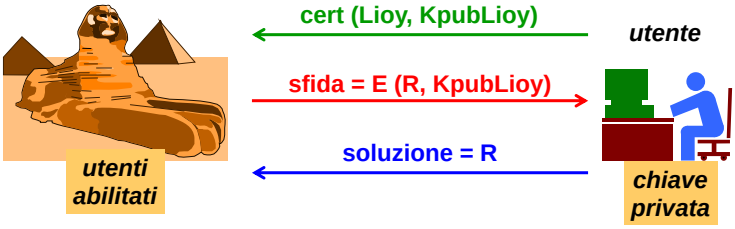


Mike intende autenticarsi a Bob fingendosi Alice, senza conoscere il segreto condiviso (cioè la chiave comune K_{AB}):

1. Mike inizia lo scambio, inviando l'UID di Alice insieme a una sfida S_A;
2. Bob invia la soluzione alla sfida S_A, insieme a una sfida S_B;
3. Mike inizia un secondo scambio, inviando l'UID di Alice insieme alla sfida S_B di cui non conosce la soluzione;
4. Bob invia la soluzione alla sfida S_B (insieme a una sfida S_C);
5. Mike invia la soluzione appena ricevuta alla sfida S_B del primo scambio, calcolata non da lui ma da Bob.

4.5 Sistemi a sfida asimmetrici

Ciò che l'utente conosce è la sua chiave privata: l'utente dimostra di conoscere la chiave senza trasmetterla, ma usandola per effettuare un calcolo. La chiave privata non è condivisa: il server può compiere la verifica usando la chiave pubblica dell'utente.



Procedura di autenticazione

1. identificazione: il client invia il proprio certificato a chiave pubblica;
2. richiesta di una prova: il server:
   (a) genera un numero R nonce;
   (b) cripta il numero generato R usando la chiave pubblica presa dal certificato;
   (c) invia il numero criptato come sfida;
3. prova: il client:
   (a) decripta il numero ricevuto usando la propria chiave privata;
   (b) trasmette il numero in chiaro come soluzione alla sfida;
4. controllo: il server confronta la prova ricevuta con il numero generato R:

$$\text{prova} = R?$$

Vantaggi

- no attacchi di sniffing: non è possibile risalire alla chiave privata dell'utente;
- no attacchi replay: la sfida contiene un numero nonce;
- no riservatezza: nessuna informazione riservata è memorizzata sul server.

Svantaggio complessità computazionale al lato client

4.5.1 Possibili errori di implementazione

- **lista degli utenti autorizzati non protetta**: un attaccante potrebbe aggiungervi il proprio nome, così da accedere usando la propria chiave:
  - soluzione: occorre garantire l'integrità e l'autenticità della lista degli utenti autorizzati memorizzata sul server;
- **certificato falso**: un attaccante potrebbe creare un certificato falso che associa la sua chiave pubblica con l'identità di uno degli utenti autorizzati, così da accedere usando la propria chiave:
  - soluzione: occorre acquisire tutti i certificati delle CA lungo il percorso di certificazione dalla CA emettitrice fino a una CA radice fidata, al fine di verificare la validità della firma digitale del certificato a chiave pubblica ricevuto;
- **name constraint violato**: un attaccante potrebbe convincere o manipolare una CA fidata ad emettere certificati al di fuori del proprio dominio (ad es. la CA del Politecnico di Milano sta certificando un utente del Politecnico di Torino):

  - soluzione: occorre acquisire il certificato della CA emettitrice, al fine di verificare che il nome dell'utente certificato rientri nel dominio dei nomi fidati (estensione "name constraints" nel certificato della CA[4]);

- **firma inconsapevole** (con chiavi RSA): se un attaccante manda come sfida il digest in chiaro di un documento, il client restituirà il digest "decriptato" con la sua chiave privata che è equivalente al digest firmato con la sua chiave privata ⇒ il client ha inconsapevolmente firmato il documento:[5]

  - soluzione: occorre usare due chiavi diverse per firma e per criptazione (estensione "key usage" nel certificato dell'utente[6]).

4.6 Autenticazione basata su password usa e getta

> **password usa e getta** una semplice tecnica di autenticazione in cui ogni password è usata solo una volta come informazione di autenticazione che verifica un'identità [...]

Ciò che l'utente conosce è una **password usa e getta** (OTP, dall'inglese "One-Time Password"), cioè non ripetibile: è valida solamente per un singolo accesso.

I sistemi OTP possono essere:

- asincroni (sez. 4.6.1): la password non dipende dal tempo (ad es. S/KEY: sez. 4.7);
- sincroni (sez. 4.6.2): la password dipende dal tempo (ad es. SecurID: sez. 4.8).

Vantaggi

- no attacchi di sniffing: il segreto dell'utente non viene trasmesso;
- no attacchi replay: la password è valida solamente per un singolo accesso.

Svantaggi

- utilizzo manuale: le password solitamente devono essere inserite da un essere umano;
- utilizzo scomodo: è scomodo inserire una password diversa a ogni accesso, soprattutto se gli accessi sono periodici (ad es. posta elettronica);
- sicurezza: le password generate da una macchina sono pseudo-casuali;
- provisioning: l'utente non può ricordarsi tutte le password possibili.

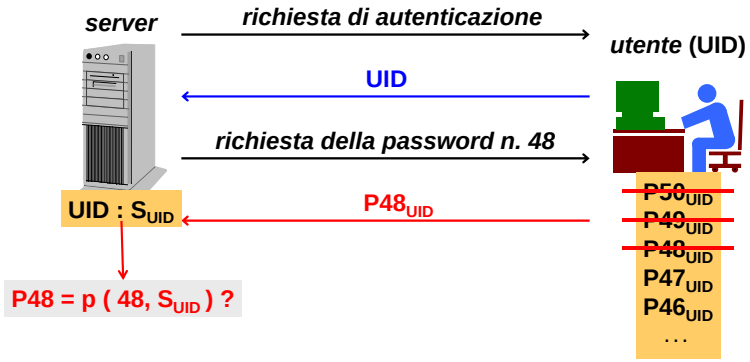
4.6.1 Sistemi OTP asincroni

La password è presa da una lista di password usa e getta inizialmente fornita al client dal server, che la genera a partire da un segreto non condiviso con l'utente.

[4] Si veda la sezione 3.2.2.

[5] La firma digitale in teoria richiederebbe la criptazione del digest, ma con chiavi RSA l'operazione di criptazione usando la chiave privata è identica all'operazione di decriptazione usando la stessa chiave privata, in quanto entrambe le operazioni consistono nell'elevamento a potenza per lo stesso esponente privato (vale una considerazione analoga per la chiave pubblica).

[6] Si veda la sezione 3.2.2.

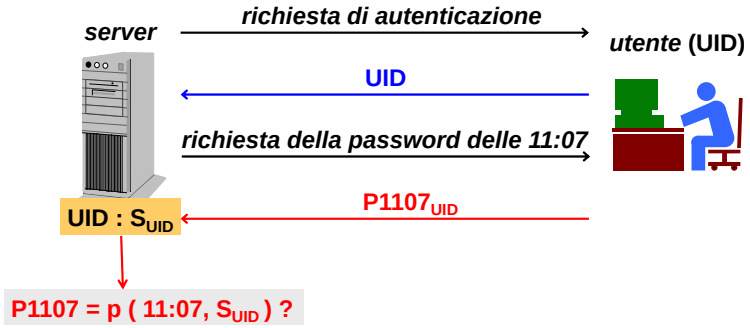


Procedura di autenticazione

1. identificazione: il client invia il proprio UID;
2. richiesta di una prova: il server invia un numero n nonce;
3. prova:
   (a) l'utente cerca la password alla posizione n-esima nella lista e la inserisce;
   (b) il client trasmette la password in chiaro;
4. controllo: il server:
   (a) calcola la password n-esima a partire dal segreto S_{UID} dell'utente;
   (b) confronta la prova ricevuta con la password calcolata:

$$\text{prova} = p\,(n, S_{\text{UID}})?$$

4.6.2 Sistemi OTP sincroni



La password è calcolata al volo dal server e dal client a partire dalla data e ora corrente, insieme al segreto condiviso dell'utente.

Procedura di autenticazione

1. identificazione: il client invia il proprio UID;
2. richiesta di una prova: il server chiede una password;
3. prova: il client:
   (a) calcola la password con la funzione di generazione p a partire dal segreto S_{UID} dell'utente e dalla data e ora corrente (ad es. tramite un token);

(b) trasmette la password in chiaro;

4. controllo: il server:
   (a) calcola la password con la funzione di generazione p a partire dal segreto S_{UID} dell'utente e dalla data e ora corrente;
   (b) confronta la prova ricevuta con la password calcolata:

$$\text{prova} = p\,(\text{data e ora}, S_{\text{UID}})?$$

Svantaggi

- costo: dovuto agli autenticatori hardware;
- sincronizzazione: poiché le password sono generate in base alla data e ora, è importante la sincronizzazione temporale tra il client e il server;
- attacchi DoS: un attaccante potrebbe indurre il server a bloccare l'account dell'utente legittimo effettuando un certo numero di tentativi di accesso falliti consecutivi:
  - palliativo: si possono introdurre dei ritardi crescenti tra i tentativi di accesso per far "stancare" l'attaccante;
- attacchi con tecniche di social engineering: un attaccante potrebbe telefonare, fingendosi l'utente legittimo, per denunciare lo smarrimento del token e chiedere l'inizializzazione di uno nuovo.

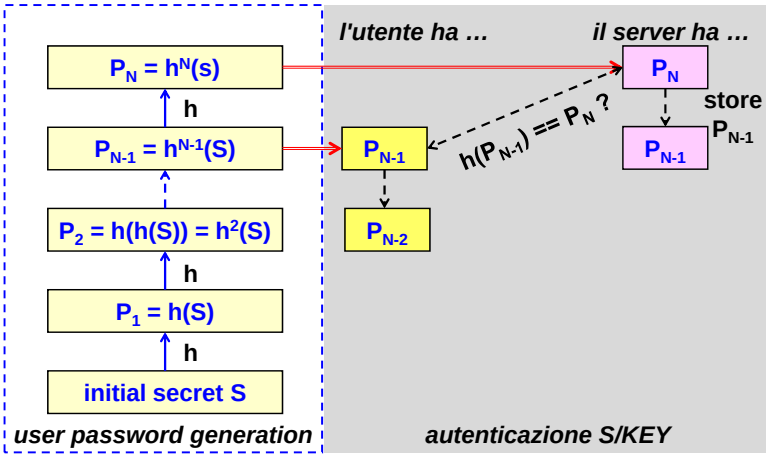
4.6.3 Modi di provisioning

Come fornire la password usa e getta all'utente?

- per client insicuri (ad es. internet point) o client privi di capacità di calcolo autonome (ad es. terminale):
  - foglio di carta su cui è scritta la lista delle password pre-calcolate dal server;
  - autenticatore hardware (sez. 4.8.2): è in grado di replicare lo stesso calcolo eseguito dal server;
- per client sicuri con capacità di calcolo autonome (ad es. dispositivo personale dell'utente):
  - autenticatore software (sez. 4.8.2): un'applicazione ad hoc, installata sul client dell'utente, in grado di replicare lo stesso calcolo eseguito dal server.
    L'applicazione può essere eventualmente integrata nel software (ad es. Telnet, FTP) o nell'hardware (ad es. router) di comunicazione per un'autenticazione automatica senza l'intervento manuale dell'utente.

4.7 Sistema S/KEY

Il sistema **S/KEY** è stato il primo sistema OTP, inventato nei Bell Labs. Il suo rilascio nel pubblico dominio (RFC) ha consentito la nascita di diverse varianti, tra cui delle implementazioni commerciali con autenticatori.



Procedura di autenticazione

1. il sistema S/KEY sul client chiede all'utente di inserire una passphrase PP segreta di almeno 8 caratteri, per semplificare la generazione delle password;
2. il sistema S/KEY concatena la passphrase con un seme inviato in chiaro dal server, formando il segreto $S \Rightarrow$ la stessa passphrase può essere usata per server diversi cambiando il seme;
3. il sistema S/KEY calcola N password applicando ripetutamente una funzione di hash h sul segreto S, e le memorizza:

$$P_1 = h(S)$$
$$P_2 = h(P_1) = h(h(S))$$
$$\dots$$

   Originariamente era usato l'algoritmo MD4 (è possibile la scelta di un algoritmo più robusto): ogni password è lunga 64 bit ed è estratta da un hash lungo 128 bit;
4. il sistema S/KEY inizializza il server di autenticazione con l'ultima password P_N: questa password non verrà mai usata direttamente per l'autenticazione, ma solo indirettamente per verificare le altre password;
5. il server, al primo accesso, chiede la password P_{N-1};
6. il server verifica se la password P_{N-1} è corretta usando la password P_N:

$$h(P_{N-1}) = P_N?$$

7. se la password P_{N-1} è corretta, il server la memorizza per poter verificare la password P_{N-2} al secondo accesso.

Vantaggi

- no attacchi di sniffing: le funzioni di hash non sono invertibili $\Rightarrow$ l'attaccante non può ricostruire la catena, a meno che non venga a conoscenza del segreto dell'utente;
- no attacchi replay: le password sono usa e getta;
- nessun segreto sul server: il server non ha bisogno di conoscere il segreto dell'utente: solo il client conosce tutte le password $\Rightarrow$ non c'è il problema della memorizzazione di informazioni sensibili sul server;
- semplicità d'uso: ogni password può essere facilmente inserita dall'utente come sequenza di 6 parole inglesi corte, convertite in bit grazie a un dizionario di 2048 parole.

4.8 RSA SecurID

token un oggetto di dato o un dispositivo fisico usato per verificare un'identità in un processo di autenticazione

Il sistema **SecurID** è stato inventato e brevettato da RSA.

Ciò che l'utente possiede è un autenticatore, chiamato **token**:

- memorizza il segreto condiviso dell'utente in modo sicuro;
- è in grado di generare password usa e getta dipendenti dal tempo (meccanismo OTP sincrono):

$$P_{\mathrm{UID}}(t) = h(S_{\mathrm{UID}}, t)$$

Il sistema è basato su un algoritmo di hash proprietario e segreto ⇒ questo è un esempio di security through obscurity: non si può avere la certezza che esso sia effettivamente robusto contro possibili attacchi.

4.8.1 Protocollo di autenticazione

SecurID è un sistema di autenticazione a due fattori:

- **token-code**: un codice di accesso, formato da 8 cifre, casuale e non ripetibile, calcolato ogni 60 secondi dal token in funzione del segreto dell'utente (seme) e della data e ora corrente;
- **PIN** associato al token: se qualcuno ruba il token, non può autenticarsi a nome del legittimo proprietario.

Una volta verificata l'identità dell'utente in base al nome utente (UID) e al PIN, il server calcola il token-code TC_0 basato sul minuto corrente e lo confronta con il token-code ricevuto. Tuttavia client e server client e server potrebbero non essere sincronizzati ⇒ se la prima verifica fallisce, il server calcola anche i token-code relativi al minuto precedente (TC_{-1}) e successivo (TC_{+1}). Dopo un certo numero (predefinito: 10) di tentativi di autenticazione falliti consecutivi, l'account dell'utente viene bloccato ⇒ ciò rende possibile gli attacchi DoS.

Come ulteriore misura di sicurezza, ad ogni autenticatore viene associato un **duress code**: se un cattivo sta costringendo l'utente ad autenticarsi sotto minaccia, egli può digitare il duress code come PIN:

- l'autenticazione sembra andare a buon fine;
- le operazioni eseguite sono applicate per finta;
- viene lanciato un allarme all'amministratore.

Ogni singolo token può avere tre diverse chiavi di accesso per essere usato con tre server diversi.

4.8.2 Autenticatori

Autenticatori hardware

Gli autenticatori hardware sono usati per:

- client insicuri (ad es. internet point);
- client privi di capacità di calcolo autonome (ad es. terminale).

Gli autenticatori hardware si differenziano in quattro categorie:

- **SecurID Card**: carta classica che semplicemente mostra il token-code corrispondente alla data e ora attuale, ma il PIN deve inserito sulla tastiera di un computer (pericoloso su client insicuri);

- **SecurID PinPad**: dispone di un tastierino integrato con cui l'utente può inserire il PIN; se il PIN inserito è corretto il dispositivo genera un token-code*, ossia una versione del token-code che incorpora già il PIN ⇒ in rete vengono trasmessi solo il nome utente e il token-code*;
- **SecurID Key Fob**: dispositivo in formato portachiavi, così l'utente se lo porta sempre con sé;
- **SecurID Key Fob and smart-card**: incorpora tutte le funzionalità dei dispositivi precedenti.

Svantaggio Poiché l'orologio al quarzo interno è soggetto nel tempo a una deriva intrinseca (al massimo 15 secondi all'anno), un autenticatore hardware ha una validità massima pari a 4 anni e deve essere sostituito.

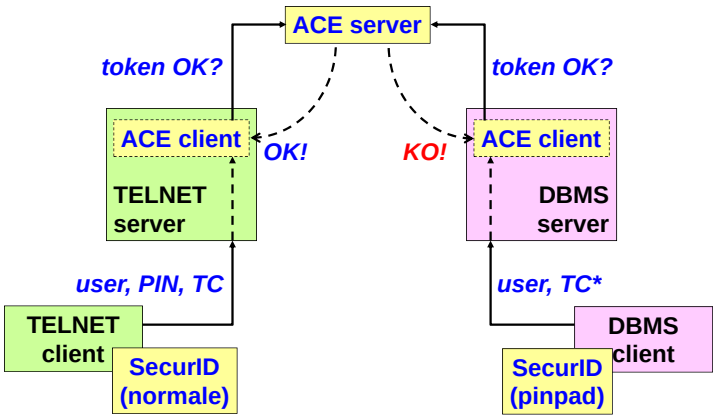
Autenticatori software

Gli autenticatori software sono installati su client sicuri e con capacità di calcolo autonome (ad es. dispositivo personale dell'utente), e implementano in software il medesimo algoritmo di calcolo degli autenticatori hardware con l'obiettivo di ridurre i costi.

Svantaggio La sincronizzazione degli orologi diventa molto rilevante: la data e ora non è più data da un orologio al quarzo ma è fornita dal sistema operativo sul dispositivo dell'utente:

- computer: la data e ora può essere sincronizzata via Internet tramite il Network Time Protocol (NTP);
- telefonino: può prendere la data e ora direttamente dalla rete cellulare.

4.8.3 Architettura di autenticazione



Oggi un'azienda non ha più un singolo server con cui l'utente interagisce direttamente tramite il protocollo di autenticazione, ma più server che devono condividere la medesima forma di autenticazione: l'intera gestione dell'autenticazione è centralizzata su un singolo server di autenticazione, a cui tutti i server applicativi si rivolgono per verificare l'identità dell'utente:

- client applicativo: interagisce direttamente con un server applicativo tramite un protocollo applicativo (ad es. Telnet, SQL), e si può autenticare per mezzo di vari autenticatori SecurID (ad es. Card, PinPad);
- server applicativo: è installato un software, chiamato **client Access Control Engine** (ACE), che contatta, attraverso un canale criptato, il server ACE ogniqualvolta è necessario verificare l'identità dell'utente;

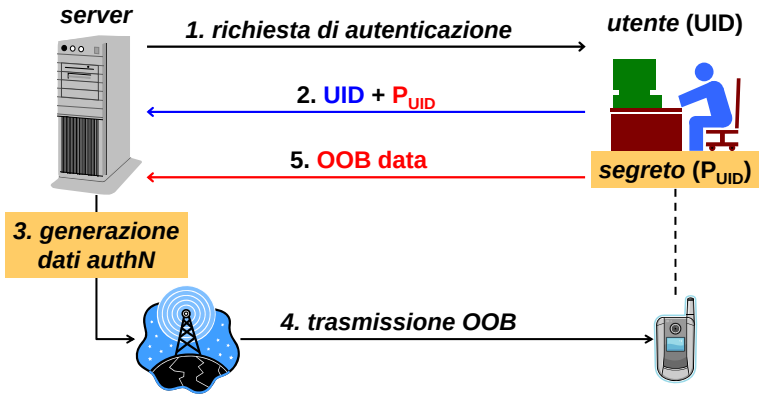
- server di autenticazione: è installato un software, chiamato **server ACE**, che è in grado di interpretare le informazioni di autenticazione memorizzando/calcolando gli elenchi, i token-code, i segreti e i PIN degli utenti.

Vantaggi

- l'utente non deve ricordarsi password diverse per accedere a servizi diversi;
- il server ACE include anche un'interfaccia SQL per l'accesso a una base di dati che memorizza i dati degli utenti, utile per una vecchia organizzazione con una base di dati preesistente.

Svantaggio Tutti i server applicativi devono fidarsi del server di autenticazione.

4.9 Autenticazione out-of-band



L'autenticazione **out-of-band** (OOB) è basata su nome utente e password, più alcuni dati aggiuntivi inviati su un altro canale di comunicazione (ad es. via SMS) fornendo così un livello di sicurezza aggiuntivo.

4.10 Sistemi biometrici

> **autenticazione biometrica** un metodo di generazione delle informazioni di autenticazione di una persona digitalizzando le misure di una caratteristica fisica o comportamentale [...]

Come essere certi di stare interagendo con un essere umano e non con un programma per computer (ad es. che invia una password memorizzata in un file)?

- **tecniche CAPTCHA** (acronimo di "Completely Automated Public Turing test to tell Computers and Humans Apart"): all'utente viene chiesto di leggere qualcosa che solo un umano è in grado di interpretare correttamente (ad es. immagini di caratteri distorti, suoni disturbati);
- **tecniche biometriche**: misurano una caratteristica biologica dell'utente (ad es. impronte digitali).

I sistemi biometrici si affidano al fatto che ogni individuo ha alcune caratteristiche biologiche uniche:

- impronte digitali;
- voce;

- scansione della retina;
- scansione della pupilla;
- scansione delle vene delle mani.

I sistemi biometrici vanno bene per un uso locale, non su Internet: se qualcuno ruba i dati biometrici, la vittima non potrà cambiare la sua caratteristica biologica (a differenza di un sistema di autenticazione basato su password).

Problemi

- accettazione psicologica: gli utenti hanno il timore di essere schedati o di essere danneggiati da tecnologie invasive;
- accuratezza: i sistemi biometrici hanno un certo margine di errore ineliminabile;
- interoperabilità: ogni dispositivo offre le proprie interfacce agli sviluppatori delle applicazioni ⇒ servono degli standard API/SPI.

4.10.1 FAR e FRR

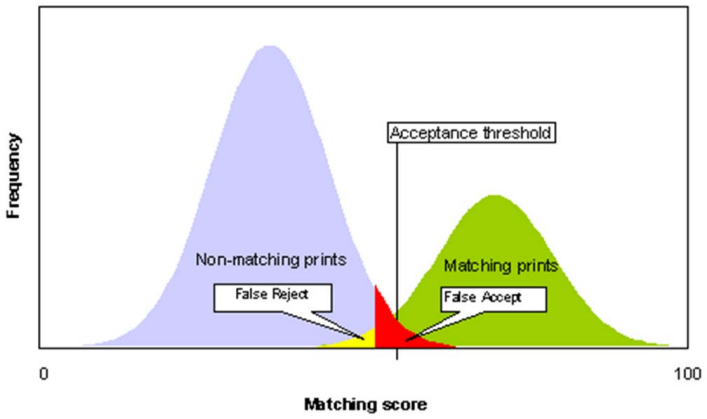


Figura 4.4: La zona di sovrapposizione delle curve gaussiane è la zona di incertezza.

Le caratteristiche fisiche di un individuo sono variabili:

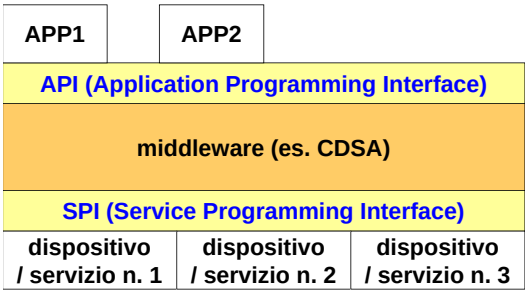
- una ferita su un dito può influenzare l'impronta digitale;
- l'emozione può rendere la voce tremante;
- alcol e droga possono dilatare i vasi oculari.

La qualità di un sistema biometrico si misura in base a due parametri statistici, che dipendono dal costo del dispositivo:

- **False Acceptance Rate** (FAR): misura quante volte l'autenticazione riesce quando l'utente non è quello registrato (falso positivo);
- **False Rejection Rate** (FRR): misura quante volte l'autenticazione fallisce quando l'utente è quello registrato (falso negativo).

Il posizionamento della **soglia di incertezza** dipende dal contesto: nelle applicazioni militari, la soglia di accettazione deve essere posizionata in modo da rifiutare sempre quando ci si trova nella zona di incertezza, anche causando molti falsi negativi.

4.10.2 CDSA



È in corso lo sviluppo di una API/SPI standard e unitaria, basata sul sistema CDSA:

- Application Programming Interface (API): offre agli sviluppatori delle applicazioni un'interfaccia comune per comunicare con dispositivi biometrici anche di produttori diversi;
- middleware (ad es. CDSA): mappa automaticamente le chiamate API alle chiamate SPI;
- Service Programming Interface (SPI): offre ai produttori dei dispositivi biometrici un'interfaccia comune per comunicare con le applicazioni anche di sviluppatori diversi.

4.11 Kerberos

Kerberos è un sistema di autenticazione sviluppato al MIT come parte del progetto Athena.
A ogni utente sono associati:

- **credenziale**: identifica univocamente l'utente ed è nella forma:
  user.instance@realm
  - **user**: è il nome utente (UID);
  - **instance**: è il ruolo dell'utente, cioè i privilegi di cui dispone (ad es. utente semplice, amministratore di sistema);
  - **realm**: è il dominio di Kerberos, ossia l'insieme di sistemi che usano Kerberos come sistema di autenticazione;
- **password**: è il segreto condiviso tra l'utente e il server di autenticazione (un utente può usare la stessa password per più instance).

4.11.1 Formato dei dati

(a) ticket *(b) autenticatore*

Figura 4.5: Formati di dato in Kerberos versione 4.

Il **ticket** è una struttura dati, generata da una terza parte fidata (TTP) e criptata con la chiave K_S del server a cui è destinato, con cui il client si autentica al server a cui il ticket è destinato:

- `server-id`: l'ID del server a cui il ticket è destinato;
- `client-id`: la credenziale dell'utente;
- `client-address`: l'indirizzo IP del client;
- `timestamp`: la data e ora in cui il ticket è stato generato;
- `life` (max 21 ore): la durata del ticket, variabile a seconda del compito per cui è stato rilasciato, al termine della quale il ticket diventa non più valido;
- $K_{S,C}$: la **chiave di sessione** simmetrica con cui è criptato l'autenticatore che accompagna il ticket; non è negoziata tra il client e il server, ma è generata al volo dalla TTP.

Il ticket costituisce, per il server a cui è destinato, la prova di autenticazione del client:

- la TTP che lo ha generato garantisce che il client ha dimostrato la sua identità;
- il ticket è fornito dalla TTP al client criptato con una chiave che solo il client legittimo conosce;
- il ticket è criptato con la chiave del server a cui è destinato $\Rightarrow$ non può essere stato modificato dal client (è una **struttura opaca**, una sorta di blob binario).

L'**autenticatore** è una struttura dati, generata dal client e criptata con la chiave di sessione $K_{S,C}$, che il client invia insieme al ticket:

- `client-id`: la credenziale dell'utente;
- `client-address`: l'indirizzo IP del client;
- `timestamp`: la data e ora in cui l'autenticatore è stato generato.

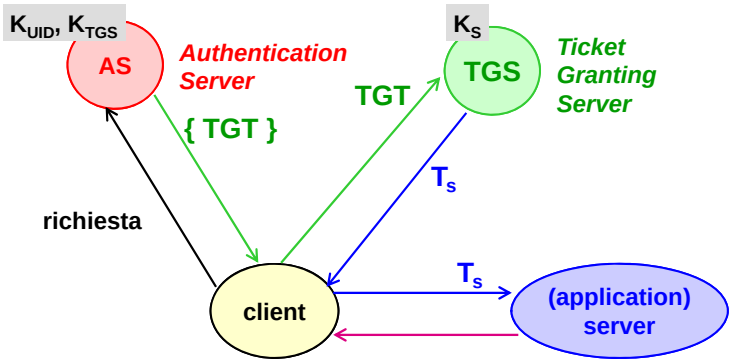
Misure di sicurezza

- contro attacchi di sniffing: tutte le comunicazioni sono criptate con chiavi simmetriche mai trasmesse in chiaro:
  - la password dell'utente e le chiavi dei server non sono mai trasmesse, ma sono solo usate localmente come chiavi di crittografia simmetrica;
  - la chiave di sessione è trasmessa dalla TTP al client in forma criptata;
  - la chiave di sessione è trasmessa dal client al server in forma criptata, contenuta all'interno del ticket;
- contro attacchi replay: l'autenticatore ha lo scopo di proteggere da attacchi replay:
  - il ticket e l'autenticatore sono legati al client tramite il suo indirizzo IP;
  - il ticket e l'autenticatore sono legati tra di loro tramite la chiave di sessione;
  - l'autenticatore ha una durata molto breve: il timestamp contenuto nell'autenticatore non può essere troppo "vecchio", e non può essere modificato perché è criptato;
  - l'autenticatore può essere usato una volta sola: è criptato con una chiave di sessione che può essere usata una volta sola.

Cambiamenti nella versione 5

- algoritmo di crittografia: la scelta dell'algoritmo di crittografia simmetrico non è più vincolata al solo DES;
- durata dei ticket: i ticket possono avere durata illimitata, perché vengono scritte esplicitamente le date e ore di inizio e fine validità (anche se è consigliato limitare sempre la durata dei ticket);
- autenticazione inter-realm: un ticket emesso all'interno di un realm diventa valido anche in un altro realm;
- ticket forwardable: anche un altro client con un altro indirizzo IP può usare il ticket;
- ticket estesi: nel ticket possono essere inserite delle informazioni addizionali che dipendono dal fornitore del sevizio.

4.11.2 Procedura di autenticazione

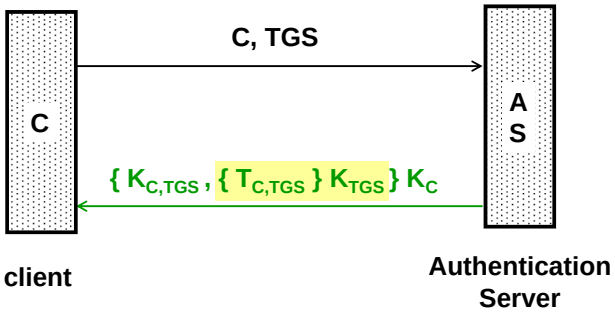


Kerberos distingue tre entità dal punto di vista logico:

- client: a ogni accesso l'utente digita localmente la sua password K_{UID};
- **Authentication Server** (AS): ha il compito di:
  - memorizzare in modo sicuro una copia delle password K_{UID} di tutti gli utenti e della chiave K_{TGS} del TGS;
  - verificare la credenziale del client;
  - generare il ticket TGT destinato al TGS;
- **Ticket Granting Server** (TGS): ha il compito di:
  - memorizzare in modo sicuro la sua chiave K_{TGS} e una copia delle password K_S dei server applicativi;
  - verificare il ticket TGT fornito dal client come prova di autenticazione;
  - generare il ticket T_S destinato al server applicativo;
- server applicativo: ha il compito di:
  - memorizzare in modo sicuro la sua chiave K_S;
  - verificare il ticket T_S fornito dal client come prova di autenticazione;
  - autenticarsi a sua volta al client in caso di autenticazione mutua;
  - fornire il servizio applicativo (ad es. stampante di rete, disco remoto).

L'AS e il TGS formano la **terza parte fidata** (TTP, dall'inglese "Trusted Third Party").

Richiesta del TGT

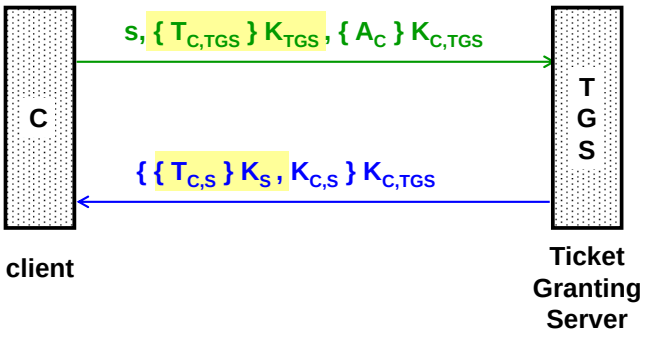


Il client chiede all'AS un ticket destinato al TGS:

1. il client invia all'AS in chiaro:
   - la sua credenziale C per identificarsi all'AS;
   - l'ID del TGS che vuole contattare;
2. l'AS invia al client:
   - il ticket $T_{C,\text{TGS}}$:
     - è criptato con la chiave K_{TGS} del TGS a cui il ticket è destinato: il ticket è opaco per il client;
     - contiene una copia della chiave di sessione $K_{C,\text{TGS}}$ tra il client e il TGS: il TGS deve conoscere la chiave di sessione per poi poter decriptare l'autenticatore;
   - la chiave di sessione $K_{C,\text{TGS}}$ tra il client e il TGS: il client deve conoscere la chiave di sessione per poi poter criptare l'autenticatore;

   entrambi criptati con la chiave K_C del client: solo il client legittimo conosce la chiave per decriptarli.

Richiesta del ticket



Il client usa il ticket fornito dall'AS per chiedere al TGS un ticket destinato al server applicativo:

3. il client invia al TGS:
   - l'ID s, in chiaro, del server applicativo che vuole contattare;
   - il ticket $T_{C,\text{TGS}}$ che ha ricevuto dall'AS, criptato con la chiave K_{TGS} del TGS: costituisce per il TGS la prova che il client possiede la chiave K_C, con cui ha potuto decriptare il ticket $T_{C,\text{TGS}}$ (autenticazione del client);

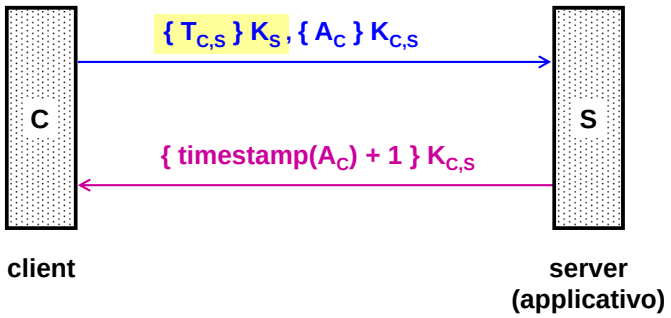
- l'autenticatore A_C, criptato con la chiave di sessione $K_{C,\text{TGS}}$ che ha ricevuto dall'AS: potrà essere decriptato dal TGS prendendo la chiave di sessione $K_{C,\text{TGS}}$ dal ticket $T_{C,\text{TGS}}$;

4. il TGS invia al client:

   - il ticket $T_{C,S}$:
     - è criptato con la chiave K_S del server applicativo a cui il ticket è destinato: il ticket è opaco per il client;
     - contiene una copia della chiave di sessione $K_{C,S}$ tra il client e il server applicativo: il server applicativo deve conoscere la chiave di sessione per poi poter decriptare l'autenticatore;
   - la chiave di sessione $K_{C,S}$ tra il client e il server applicativo: il client deve conoscere la chiave di sessione per poi poter criptare l'autenticatore;

   entrambi criptati con la chiave di sessione $K_{C,TGS}$: solo il client autenticato all'AS conosce la chiave per decriptarli.

Uso del ticket



Il client usa il ticket fornito dal TGS:

5. il client invia al server applicativo:

   - il ticket $T_{C,S}$ che ha ricevuto dal TGS, criptato con la chiave K_S del server applicativo;
   - l'autenticatore A_C, criptato con la chiave di sessione $K_{C,S}$ che ha ricevuto dal TGS: potrà essere decriptato dal server applicativo prendendo la chiave di sessione $K_{C,S}$ dal ticket $T_{C,S}$;

6. il server applicativo invia al client il timestamp +1 dell'autenticatore A_C, criptato con la chiave di sessione $K_{C,S}$: costituisce per il client la prova che il server applicativo possiede la chiave K_S, con cui ha potuto decriptare il ticket da cui ha preso la chiave con cui decriptare l'autenticatore (autenticazione del server).

4.11.3 Vantaggi

- versatilità: molti servizi possono essere "kerberizzati", cioè adattati ad usare il meccanismo dei ticket: K-POP, K-NFS (per dischi di rete), K-LPD (per stampanti di rete), K-telnet, K-ftp, K-dbms;
- accesso unico per tutti i servizi kerberizzati: un ticket può essere usato per più servizi kerberizzati;
- connessioni intermittenti: il ticket rimane valido anche se il client si disconnette dalla rete e si riconnette ad essa, evitando continue richieste di accesso;

- supporto commerciale: Microsoft ha adottato Kerberos*, un'implementazione proprietaria di Kerberos, a partire da Windows 2000.

4.11.4 Problemi

- memorizzazione dei segreti sul server: l'AS memorizza in chiaro tutte le password degli utenti e la chiave del TGS, mentre il TGS memorizza il chiaro le chiavi dei server applicativi;
- attacchi IP spoofing: il ticket e l'autenticatore sono legati al client tramite il suo indirizzo IP;
- attacchi DoS: un attaccante può sovraccaricare un server Kerberos effettuando molte richieste di ticket;
- sincronizzazione degli orologi: l'autenticatore può non essere più valido se il tempo impiegato per andare dal client al server è troppo lungo:
  - LAN: è utile perché in caso di attacco è importante poter ricostruire che cosa è successo e correlare gli eventi a scopo di auditing;
  - WAN: i ritardi possono essere troppo elevati a causa delle distanze più lunghe ⇒ Kryptonight è un'implementazione di Kerberos sviluppata da IBM che non ha più bisogno dei timestamp;
- accesso remoto: se l'utente si collega da remoto (ad es. casa), la password deve essere trasmessa in chiaro attraverso la rete fino alla sua postazione kerberizzata ⇒ sono necessari un canale criptato o altri meccanismi di autenticazione.

4.11.5 SSO

Single Sign-On (SSO) fornire all'utente un'unica "credenziale" con cui autenticarsi per tutte le operazioni su qualunque sistema

Kerberos è un esempio di un concetto più generale, il **Single Sign-On** (SSO):

- fittizio: i singoli servizi richiedono autenticazioni indipendenti con password diverse:
  - password wallet: chiede una singola password globale all'utente per accedere al database delle password, e poi fornisce la password corretta a ogni servizio che richiede l'autenticazione in modo trasparente all'utente;
- reale: tutti i servizi condividono effettivamente lo stesso sistema di autenticazione:
  - tecniche di autenticazione multi-applicazione (ad es. sistemi a sfida asimmetrici, Kerberos): tutti i servizi sono in grado di riconoscere la credenziale unica dell'utente (ad es. certificato a chiave pubblica, ticket Kerberos);
  - SSO multi-dominio: un servizio accetta la credenziale (ad es. token SAML) di un servizio in un altro dominio (ad es. Google).

4.12 Interoperabilità dell'autenticazione

Le implementazioni proprietarie causano problemi di interoperabilità tra soluzioni di autenticazione di fornitori diversi.

Negli ultimi anni è nata un'iniziativa chiamata **OATH** che mira a realizzare l'interoperabilità dei sistemi OTP, a sfida simmetrici e a sfida asimmetrici, definendo degli standard per il protocollo client-server e per il formato dei dati sul client. Si ottiene l'**autenticazione forte universale** quando tutti gli utenti possono usare dispositivi di qualsiasi produttore, che soddisfino le specifiche OATH, per qualunque servizio di rete.

Sono già state pubblicate alcune specifiche sotto forma di RFC:

- HMAC OTP (HOTP): password usa e getta basate su digest HMAC;
- Time-based OTP (TOTP): password usa e getta basate sul tempo;
- OATH Challenge-Response Protocol (OCRA): protocollo a sfida;
- Portable Symmetric Key Container (PSKC): formato XML per trasportare chiavi simmetriche;
- Dynamic Symmetric Key Provisioning Protocol (DSKPP): protocollo di rete per la distribuzione di chiavi simmetriche da un key server.

Capitolo 5

Sicurezza delle reti IP

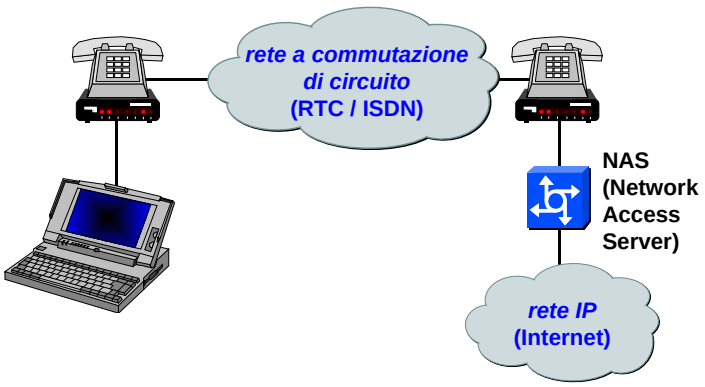
La sicurezza può essere implementata a qualunque livello della pila OSI (ad eccezione del livello di presentazione, che si limita alla conversione dei dati): qual è il livello OSI ottimale a cui implementare la sicurezza?

In generale non è possibile definire un livello ottimale, ma è necessario combinare più soluzioni a vari livelli:

- livelli superiori: si possono bloccare anche gli attacchi più raffinati:
  - \+ le decisioni sono basate su informazioni specifiche (ad es. nome utente, comandi, dati) ⇒ le decisioni sono migliori;
  - − gli intrusi possono agire indisturbati nei livelli sottostanti (ad es. attacchi DoS);
- livelli inferiori: è meglio limitarsi a bloccare solo gli attacchi più di base:
  - − i dati su cui basare le decisioni sono scarsi (ad es. indirizzo MAC, indirizzo IP) ⇒ si rischia di bloccare dei buoni e lasciar passare dei cattivi;
  - \+ è possibile "espellere" più in fretta gli intrusi.

5.1 Autenticazione per connessioni ad accesso remoto

5.1.1 Autenticazione del canale



In passato le connessioni a Internet erano ad accesso remoto: l'utente si connette attraverso un modem, che comunica direttamente con il modem dell'ISP attraverso un collegamento punto-punto su una rete a commutazione di circuito di livello data-link (cioè senza router intermedi).

Il modem dell'utente e il modem dell'ISP comunicano usando il **Point-to-Point Protocol** (PPP), che è in grado di incapsulare i pacchetti di livello 3 (ad es. IP) e di trasportarli su un collegamento punto-punto, chiamato **canale PPP**:

- fisico (ad es. Rete Telefonica Commutata [RTC], Integrated Services Digital Network [ISDN]): c'è un filo fisico tra il modem dell'utente e il modem dell'ISP su cui viaggiano le trame PPP;
- virtuale: c'è un collegamento logico (tunnel) tra il modem dell'utente e il modem dell'ISP:
  - di livello 2 (ad es. xDSL con PPPoE): le trame PPP sono incapsulate in trame Ethernet;
  - di livello 3 (ad es. VPN con L2TP): le trame PPP sono incapsulate in pacchetti UDP.

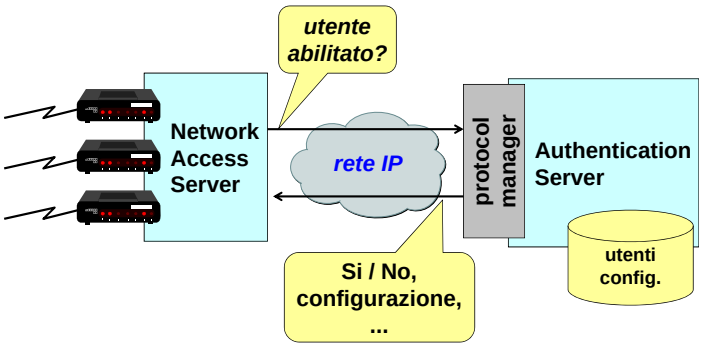
Il canale PPP viene attivato in tre fasi sequenziali:

1. connessione (ad es. tramite Link Control Protocol [LCP]): viene aperta la connessione e vengono negoziate alcuni parametri;
2. autenticazione (ad es. tramite PAP, CHAP, EAP) (facoltativa): il canale PPP di livello 2 viene autenticato, cioè l'utente mostra la sua identità;
3. incapsulamento (ad es. tramite IP Control Protocol [IPCP]): le trame PPP vengono incapsulate in pacchetti di livello 3 (ad es. IP).

Esistono principalmente tre protocolli che permettono di eseguire la fase di autenticazione del canale PPP:

- PAP (Password Authentication Protocol): il nome utente e la password sono inviati in chiaro;
- CHAP (Challenge Handshake Authentication Protocol): usa un sistema di autenticazione a sfida simmetrico, basato sulla password dell'utente;
- EAP (sez. 5.2): il protocollo è indipendente dal sistema di autenticazione esterno (ad es. OTP, a sfida, TLS).

5.1.2 Verifica dell'autenticazione



Il **Network Access Server** (NAS) dell'ISP è il gateway verso la rete esterna (ad es. Internet): il suo compito principale è tradurre i segnali analogici (telefonici) in bit (pacchetti IP).

In aggiunta, il NAS deve fornire le tre funzioni di sicurezza conosciute come "le tre A":

- **Authentication**: verificare l'identità dichiarata dall'utente in base alle sue credenziali (ad es. password, OTP);
- **Authorization**: verificare quali azioni l'utente può compiere e a quali servizi può accedere;

accounting il processo di tenere traccia e/o analizzare le attività degli utenti su una rete registrando i dati chiave (ad es. quantità di tempo nella rete, servizi acceduti, quantità di dati trasferiti)

- **Accounting**: tenere traccia dell'attività dell'utente per vari scopi (ad es. addebito, audit).

Si potrebbero inserire direttamente nel NAS tutte le informazioni sugli utenti abilitati al servizio, ma se i NAS sono multipli e non vicini è necessario concentrare il database degli utenti in un singolo server, chiamato **Authentication Server** (AS), a cui tutti i NAS sono collegati tramite una rete IP locale:

1. il NAS chiede all'AS di verificare se l'utente è abilitato a connettersi alla rete;
2. l'AS restituisce l'esito della verifica, insieme in caso affermativo alla configurazione di rete dell'utente (ad es. proxy, DNS).

Ogni NAS comunica con l'AS usando uno dei protocolli seguenti:

- RADIUS (sez. 5.3): è il protocollo attualmente più diffuso;
- DIAMETER (sez. 5.4): è un'evoluzione di RADIUS;
- TACACS+ (TACACS, XTACACS): in origine era tecnicamente migliore di RADIUS, ma essendo un protocollo proprietario di Cisco è meno diffuso.

I NAS possono comunicare con l'AS usando protocolli diversi l'uno dall'altro ⇒ il **protocol manager** si occupa di supportare più protocolli di autenticazione allo stesso tempo.

5.2 EAP

Il **Extensible Authentication Protocol** (EAP) è un framework flessibile di autenticazione a livello data-link.

EAP pre-definisce tre tipi di autenticazione:

- MD5-challenge: sistema a sfida simile a CHAP e basato su keyed-digest MD5;
- OTP;
- generic token card.

EAP è estensibile: possono essere aggiunti nuovi tipi di autenticazione:

- PPP EAP TLS authentication protocol: aggiunge il supporto a TLS;
- RADIUS support for EAP: aggiunge il supporto a RADIUS.

5.2.1 Protocollo di incapsulamento

La fase di autenticazione avviene prima dell'instaurazione del collegamento di livello 3 ⇒ EAP usa un proprio protocollo di incapsulamento per i dati di autenticazione:

- è indipendente dal protocollo di livello 2: può autenticare non solo collegamenti PPP, ma anche collegamenti come 802.11 (wi-fi), 802.3 (Ethernet), 802.5 (token ring);
- richiede ACK/NACK espliciti: ogni pacchetto deve essere confermato con un ACK, altrimenti viene ritrasmesso fino a 3/4 volte;
- non implementa meccanismi a finestra di scorrimento: assume che tutti i pacchetti arrivino sempre in sequenza:

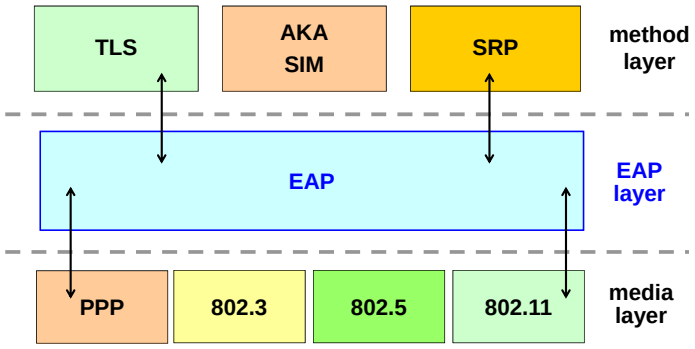
  - canale PPP fisico: il PPP garantisce che i pacchetti arrivino sempre in sequenza (singolo link);
  - canale PPP virtuale: in UDP e IP i pacchetti possono arrivare fuori sequenza (più link);
- non supporta la frammentazione: è compito dei metodi EAP gestire la frammentazione se il payload è maggiore della MTU.

5.2.2 Metodi di autenticazione

EAP assume il link fisico come insicuro: nessuna password è trasmessa in chiaro, ma i necessari servizi di sicurezza sono forniti dal metodo di autenticazione scelto:

- EAP-TLS: protocollo base per il Web;
- EAP-MD5: sistema a sfida simmetrico e basato su un keyed-digest MD5 per l'autenticazione del client (non mutua);
- EAP-TTLS: stabilisce un tunnel TLS sicuro in cui operare qualsiasi metodo EAP (anche nome utente e password);
- EAP-SRP (Secure Remote Password);
- GSS_API: generalizzazione del sistema Kerberos;
- AKA-SIM (Authenticated Key Exchange): sistema a sfida basato su dati presi dalla scheda SIM di un cellulare.

5.2.3 Architettura



Dal punto di vista architetturale si distinguono tre livelli:

- **media layer**: è il protocollo di livello 2 sottostante che trasporta le trame;
- **EAP layer**: è il protocollo di incapsulamento EAP per i dati di autenticazione;
- **method layer**: è il metodo di autenticazione EAP per il servizio di sicurezza.

L'EAP layer è una sorta di middleware: qualsiasi metodo di autenticazione può essere usato con qualsiasi protocollo di livello 2.

5.3 RADIUS

Il protocollo **Remote Authentication Dial-In User Service** (RADIUS) è attualmente lo standard de facto per l'autenticazione remota.

Caratteristiche

- implementa uno schema client-server tra NAS e AS;
- permette l'amministrazione e l'accounting degli utenti centralizzati sull'AS;
- permette l'accesso alla rete tramite:
  - porte fisiche: analogiche, ISDN, IEEE 802;
  - porte virtuali: tunnel, accessi wireless;
- giace sul protocollo UDP: usa la porta 1812 per l'autenticazione e la porta 1813 per l'accounting (esistono implementazioni non ufficiali basate sulle porte 1645 e 1646);
- estende il protocollo UDP: il pacchetto viene ritrasmesso allo scadere di un timeout;
- supporta l'autenticazione dell'utente tramite i protocolli PAP, CHAP e EAP;
- gli attributi sono nel **formato Type-Length-Value** (TLV): se un server non conosce un tipo di attributo, lo può "saltare" grazie al campo della lunghezza;
- il server RADIUS può fungere da proxy verso altri server di autenticazione: la richiesta di accesso di un utente appartenente ad un altro dominio è inoltrata al server di autenticazione non RADIUS di quel dominio, con un proprio database, delegando ad esso la parte di autenticazione:

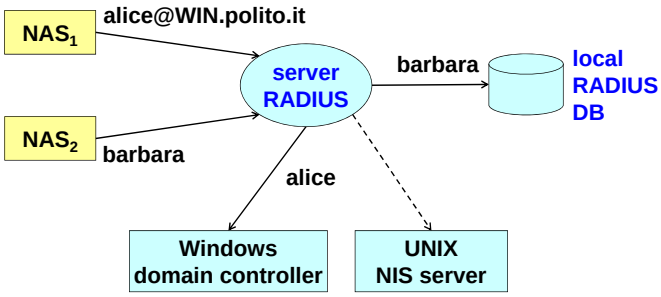
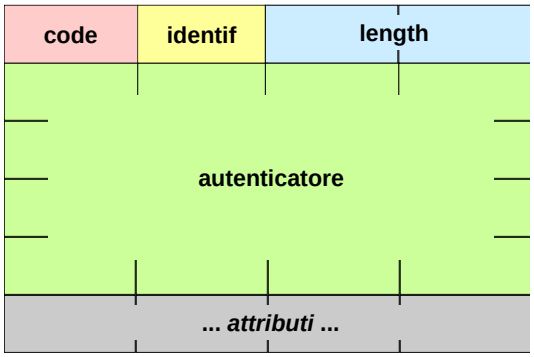


Figura 5.1: La richiesta di accesso di Alice è inoltrata al server "Windows domain controller" appartenente al dominio `WIN.polito.it`.

5.3.1 Formato dei pacchetti



Tutti i pacchetti RADIUS hanno il seguente formato:

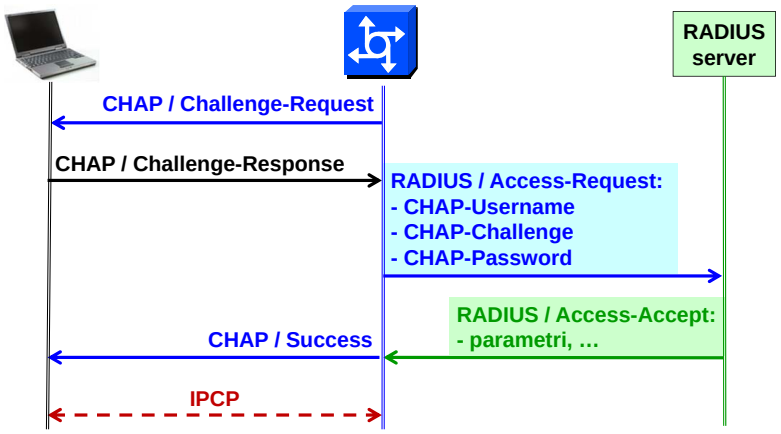
- campo Code (1 byte): identifica il tipo di pacchetto:

  - valore “Access-Request”: è la richiesta del NAS, e contiene le credenziali di accesso (ad es. nome utente e password);
  - valore “Access-Accept”: l’AS comunica che la richiesta di accesso è stata accettata, e contiene i parametri per la configurazione di rete del nodo dell’utente;
  - valore “Access-Reject”: l’AS comunica che la richiesta di accesso è stata negata (ad es. a causa di credenziali errate);
  - valore “Access-Challenge”: l’AS richiede informazioni aggiuntive all’utente (ad es. PIN, token-code, password secondaria);
- campo Identifier (1 byte): identifica la richiesta;
- campo Length (2 byte): specifica la lunghezza del pacchetto;
- campo Authenticator (16 byte): contiene l’autenticatore, ed è calcolato in due modi diversi a seconda del tipo di pacchetto:
  - **Request Authenticator**: si trova nelle richieste del NAS, ed è formato da 16 byte casuali e nonce generati dal NAS;
  - **Response Authenticator**: si trova nelle risposte dell’AS, ed è il risultato di un keyed-digest MD5:
    $$\text{md5}(\text{ code } || \text{ ID } || \text{ length } || \text{ Request Authenticator } || \text{ attributi } || \text{ chiave })$$
    * campi (code, ID, length, attributi): fornisce l’integrità del pacchetto;
    * Request Authenticator: la risposta è legata alla richiesta $\Rightarrow$ impedisce gli attacchi replay;
    * chiave: è il segreto condiviso con il NAS $\Rightarrow$ fornisce autenticazione dell’AS nei confronti del NAS;
- **attributi**:
  - attributo User-Name (tipo 1): contiene un testo generico, un Network Access Identifier (NAI) (= nome_utente[@realm]) o un distinguished name (DN);
  - attributo User-Password (tipo 2): contiene la password dell’utente offuscata (mascherata) tramite uno pseudo-algoritmo di criptazione:
    $$(\text{ password } || \text{ padding }) \oplus \text{md5}(\text{ chiave } || \text{ Request Authenticator })$$
    * chiave: è il segreto condiviso con l’AS $\Rightarrow$ fornisce autenticazione del NAS nei confronti dell’AS;
    * Request Authenticator: è casuale $\Rightarrow$ due richieste di accesso dello stesso utente avranno risultati diversi;
  - attributo Chap-Password (tipo 3): contiene la risposta dell’utente alla sfida (128 bit);
  - attributo Chap-Challenge (tipo 60): contiene la sfida generata dal NAS all’utente.

5.3.2 Esempio di autenticazione con CHAP

L’utente desidera collegarsi tramite un NAS, il quale è collegato ad un server RADIUS:

1. CHAP / Challenge-Request: il NAS invia all’utente una sfida;
2. CHAP / Challenge-Response: l’utente invia al NAS la risposta alla sfida, ma solo il server RADIUS ha la password dell’utente e può verificare se la risposta alla sfida è corretta;
3. RADIUS / Access-Request: il NAS invia al server RADIUS una richiesta di accesso contenente i seguenti attributi:
   - CHAP-Username: il nome utente fornito dall’utente;



- CHAP-Challenge: la sfida generata dal NAS all'utente;
- CHAP-Password: la risposta alla sfida fornita dall'utente;

4. RADIUS / Access-Accept: se l'utente è abilitato e la risposta alla sfida è corretta, il server RADIUS invia al NAS i parametri di configurazione di rete per l'utente;
5. CHAP / Success: il NAS "passa" all'utente i parametri di configurazione di rete;
6. IPCP: attiva il collegamento IP.

5.3.3 Analisi di sicurezza

Purtroppo RADIUS non fornisce tutte le funzionalità di sicurezza che un protocollo di autenticazione necessiterebbe per scambiare le richieste del NAS e le risposte dell'AS in modo sicuro.

Richiesta del NAS

- **sniffing**:
  - confidenzialità: la password non è criptata, ma è solo offuscata:
    - − PAP: ciò che viene offuscato è la password in chiaro ⇒ il cattivo può rubare la password, anche se con tecniche molto complicate;
    - \+ CHAP: la password non è mai trasmessa ⇒ il cattivo non può rubare la password;
    - \+ EAP: la password non è trasmessa o è trasmessa non in chiaro ⇒ il cattivo non può rubare la password;
  - − privacy: il nome utente è trasmesso in chiaro ⇒ il cattivo può sapere quando l'utente si è collegato/scollegato;
- **attacco a forza bruta** (password enumeration): il cattivo potrebbe scoprire la password dell'utente usando l'AS come oracolo:
  - \+ autenticazione: la password è offuscata anche includendo la chiave che è il segreto condiviso con il NAS ⇒ il cattivo non può fingere di essere un NAS;
  - \+ nessuna risposta: se la chiave è errata, la richiesta viene ignorata senza alcun messaggio di errore;
- **attacco DoS**: il cattivo potrebbe saturare l'AS con tante richieste:
  - \+ scalabilità: permette di passare a un AS secondario se l'AS non è disponibile ⇒ non basta saturare un singolo server.

Risposta dell'AS

- **falsificazione**: il cattivo potrebbe permettere l'accesso a quell'utente non valido o negare l'accesso a quell'utente valido:
  - + autenticazione: il Response Authenticator è un keyed-digest MD5: la chiave è il segreto condiviso con l'AS ⇒ il cattivo non può fingere di essere l'AS;
  - + integrità: il Response Authenticator è un keyed-digest MD5: il digest non corrisponderà se è avvenuta una modifica ⇒ il cattivo non può modificare la risposta dell'AS;
- **attacco replay**: il cattivo potrebbe permettere l'accesso ad un altro utente non valido o negare l'accesso ad un altro utente valido:
  - + risposta legata alla richiesta: l'AS non si limita a rispondere con un semplice sì o no, ma il Response Authenticator è calcolato anche includendo il Request Authenticator.

5.4 DIAMETER

Il protocollo **DIAMETER** è un'evoluzione di RADIUS:

- pone enfasi sul roaming tra ISP diversi: gli utenti di un ISP possono collegarsi da un altro ISP in roaming (come nella rete cellulare);
- è più attento di RADIUS alla sicurezza, perché è pensato con la sicurezza in mente:
  - RADIUS: la sicurezza è integrata nel protocollo, quindi non possono essere adottati dei nuovi meccanismi di sicurezza più robusti;
  - DIAMETER: la funzionalità principale del protocollo è staccata dalla sicurezza, obbligando ad usare un protocollo di sicurezza esterno esistente.

Non è tuttavia ancora molto usato perché:

- RADIUS funziona molto bene;
- il roaming tra ISP non è ancora così rilevante: due ISP devono portare a termine un accordo commerciale per consentire il roaming agli utenti.

Protocolli di sicurezza obbligatori L'RFC specifica di usare obbligatoriamente dei protocolli di sicurezza di livello superiore:

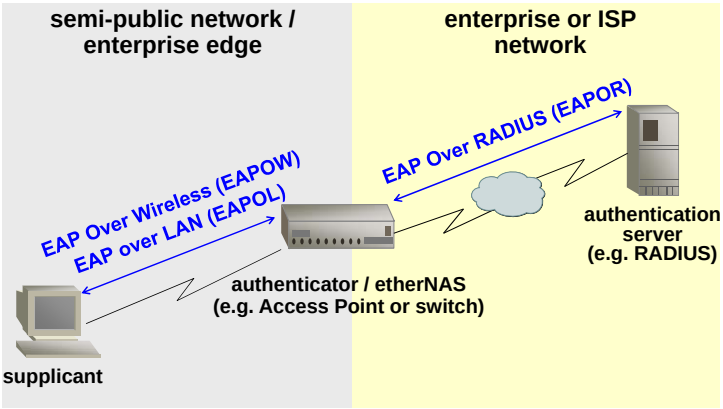
- **IPsec** (livello 3: sez. 5.9): deve essere supportato sia dai client sia dai server DIAMETER:
  - autenticazione e riservatezza: deve essere usato ESP, con algoritmi non nulli, che protegge l'intero pacchetto;
- **TLS** (livello 7: sez. 7.3.13): deve essere supportato dai server DIAMETER (e può essere supportato dai client DIAMETER):
  - autenticazione e riservatezza: deve essere supportata la combinazione RSA + RC4_128/3DES + MD5/SHA1 (e può essere usata la combinazione RSA + AES_128 + SHA1);
  - mutua autenticazione: anche il server deve autenticarsi, e il client deve avere un certificato a chiave pubblica.

5.5 IEEE 802.1x

Lo standard **IEEE 802.1x**, denominato "Port-Based Network Access Control", ha avuto molto successo sin dalle prime implementazioni Microsoft (Windows XP) e Cisco, e definisce un'architettura generalizzata che fornisce un framework per effettuare:

- **autenticazione a livello MAC** (livello 2): l'accesso alla rete è limitato ai soli utenti abilitati tramite un'autenticazione richiesta dal dispositivo di rete stesso:
  - rete cablata: l'utente è attaccato fisicamente con un cavo alla porta di uno switch ⇒ l'autenticazione non è indispensabile perché l'utente deve avere l'accesso fisico alla porta;
  - rete senza fili: l'utente si collega a distanza a un access point ⇒ l'autenticazione diventa indispensabile in quanto l'utente può essere ovunque;
- **key management**: sono necessarie delle chiavi per proteggere le comunicazioni (indispensabile per wireless):
  - può derivare chiavi di sessione da usare per autenticazione, integrità o segretezza dei pacchetti, a seconda delle caratteristiche di sicurezza desiderate per la comunicazione;
  - si limita a fornire i formati dei pacchetti per trasportare le informazioni, sfruttando algoritmi standard per la derivazione delle chiavi (ad es. TLS, SRP);
  - i servizi di sicurezza sono facoltativi: sta al client e all'access point decidere se usare solo autenticazione oppure autenticazione + criptazione.

5.5.1 Architettura



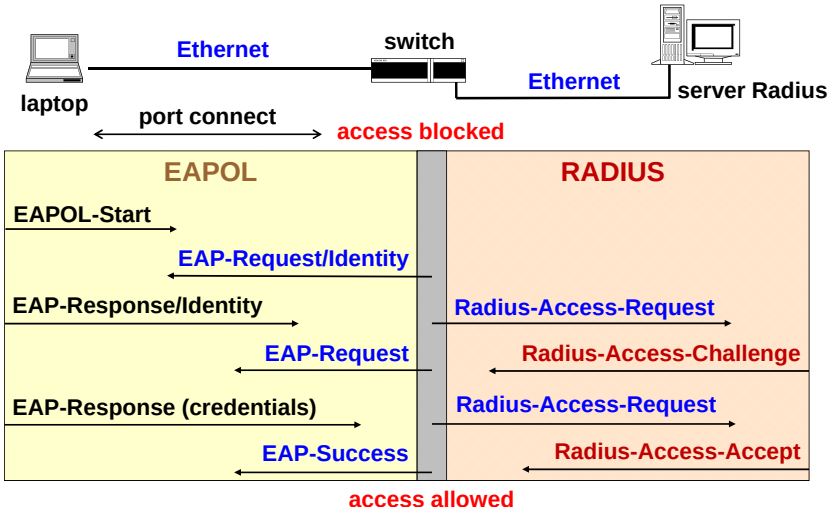
Dal punto di vista architetturale si distinguono tre entità:

- **supplicant**: è il client dell'utente che "supplica" di avere accesso alla rete;
- **authenticator** (o etherNAS): è il punto di accesso alla rete aziendale o dell'ISP, cioè lo switch o l'access point wireless, che ha lo scopo di:
  - bloccare l'accesso alla rete fino a quando il client non è stato autenticato con successo;
  - fungere da proxy verso il server durante la fase di autenticazione;
- **server di autenticazione**: è l'AS RADIUS, che ha lo scopo di accettare o rifiutare le richieste di accesso.

La richiesta di accesso del supplicant va al server di autenticazione attraverso l'authenticator:

1. il supplicant invia all'authenticator un messaggio EAP di richiesta di accesso tramite la rete semi-pubblica o aziendale di livello 2:
   - EAP Over LAN (EAPOL) se la rete è cablata;
   - EAP Over Wireless (EAPOW) se la rete è senza fili;
2. l'authenticator incapsula il messaggio EAP in RADIUS (EAP Over RADIUS [EAPOR]) e lo invia all'AS tramite la rete IP aziendale o dell'ISP.

5.5.2 Messaggi



Quando l'utente connette il suo portatile alla porta dello switch Ethernet:

1. port connect: lo switch rileva la presenza di attività elettrica sulla porta di accesso, che si trova nello stato "bloccata" predefinito da cui può uscire solo al termine di un processo di autenticazione;
2. EAPOL-Start: il software 802.1x sul client chiede l'inizio del processo di autenticazione in modo da avere accesso alla rete;
3. EAP-Request/Identity: lo switch chiede al client di mostrare la propria identità;
4. EAP-Response/Identity: il client invia le informazioni richieste sulla propria identità;
5. Radius-Access-Request: lo switch inoltra queste informazioni al server, reincapsulandole nel formato dati RADIUS;
6. Radius-Access-Challenge: il server genera una sfida per il client;
7. EAP-Request: lo switch inoltra la sfida al client, reincapsulandola nel formato dati EAP;
8. EAP-Response: il client fornisce le credenziali, contenenti il nome utente e la risposta alla sfida;
9. Radius-Access-Request: lo switch inoltra queste credenziali al server, reincapsulandole nel formato dati RADIUS;
10. Radius-Access-Accept: se la risposta alla sfida è corretta e l'utente è abilitato all'accesso alla rete, il server accetta la richiesta di accesso;

11. EAP-Success: lo switch inoltra questa risposta al client, reincapsulandola nel formato dati EAP;

12. lo switch sblocca la porta, e dà al client la configurazione di rete tramite DHCP.

A differenza dell'autenticazione con CHAP, la sfida non è generata dal NAS ma dal server RADIUS.

5.5.3 Vantaggi

802.1x non inventa nuovi protocolli, ma utilizza i protocolli già esistenti:

- conversazione diretta (a livello concettuale) tra il supplicant e l'AS: la scheda di rete (NIC) del supplicant e l'authenticator agiscono soltanto come dispositivi "passacarte";
- sfrutta il protocollo EAP per l'effettiva implementazione dei meccanismi di sicurezza:
  - è possibile utilizzare tutti i sistemi di autenticazione compatibili con EAP;
  - facilmente estensibile: il supporto a nuovi metodi di autenticazione non richiede modifiche né alla NIC né all'authenticator;
- sfrutta il protocollo RADIUS integrandosi perfettamente con le tre A:
  - accounting: l'azienda o l'ISP può registrare quando ogni utente si è connesso e disconnesso ⇒ è utile sapere chi era conesso alla rete mentre avveniva un attacco (audit).

5.6 Sicurezza del DHCP

Il DHCP è un protocollo che risiede tra il livello 2 e il livello 3, e serve a fornire la configurazione di rete a un nodo che ne è privo e desidera entrare in una rete IP.

DHCP non è un **protocollo non autenticato**: oltre al fatto che l'utente che richiede la configurazione di rete non è autenticato, il problema principale è che la risposta non è autenticata ⇒ è molto semplice attivare degli shadow server come falsi server DHCP, rendendo possibili due tipi di attacchi:

- **attacchi DoS**: al client è fornita una configurazione di rete errata e non funzionante (ad es. indirizzo DNS errato) che gli impedisce di navigare ⇒ l'utente se ne accorge e chiama l'amministratore di rete;
- **attacchi man-in-the-middle logico**: al client è fornita una configurazione di rete errata ma funzionante ⇒ l'utente non se ne accorge:
  - pacchetti in uscita: il client appartiene a una rete con netmask '/30', e il suo default gateway è lo shadow server;
  - pacchetti in entrata: le risposte possono essere intercettate definendo un NAT.

Ci sono alcuni tentativi per fornire protezione al DHCP:

- **DHCP snooping**: l'amministratore definisce a quali porte di uno switch possono arrivare delle risposte da parte di server DHCP ⇒ un attaccante non può mandare risposte DHCP collegandosi a una porta non abilitata;
- **IP guard**: vengono scartati i pacchetti con indirizzo IP sorgente non incluso tra quelli forniti tramite DHCP ⇒ limita gli attacchi di IP spoofing, ma lo switch deve memorizzare molti indirizzi IP;
- **autenticazione dei messaggi DHCP**: le risposte DHCP sono autenticate con un keyed-digest HMAC MD5 ⇒ sarebbe la soluzione ideale, ma:

– è scarsamente adottata;
– bisognerebbe usare chiavi distinte, altrimenti a un cattivo basta avere un client valido per conoscere la chiave simmetrica e usarla per un server DHCP falso.

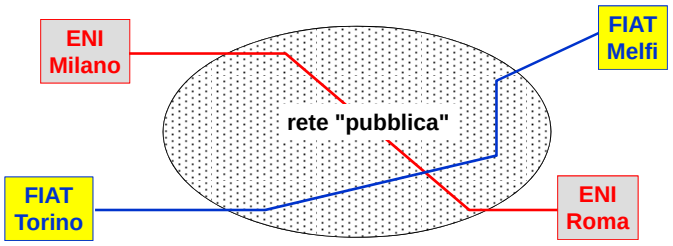
5.7 Sicurezza a livello rete

Il livello rete è il primo livello realmente interessante della pila OSI in cui implementare le funzioni di sicurezza, perché è il primo livello a collegare i nodi di elaborazione in modo end-to-end.

A livello rete si possono creare due tipi di funzioni di sicurezza:

- protezione end-to-end: il client negozia direttamente con il server le funzioni di sicurezza:
  + la protezione funziona anche se avvengono degli attacchi lungo il tunnel;
  – richiede la configurazione di tutti i client, ma gli utenti possono non essere così abili;
- protezione della rete (ad es. VPN): le funzioni di sicurezza sono sui nodi intermedi (router):
  – il client e il server sono i punti deboli della rete;
  + la protezione è trasparente agli utenti.

5.8 VPN



Virtual Private Network (VPN) una tecnica hardware e/o software per realizzare una rete privata utilizzando canali e apparati di trasmissione condivisi o comunque non fidati

Si supponga ad esempio che esista una rete pubblica (o equivalente a pubblica, ossia non di uso personale) sulla quale si vuole trasmettere il proprio traffico. Un'azienda che utilizza una rete del genere, e che vuole essere in grado di comunicare in modo sicuro con tutte le sue sedi, dovrà in qualche modo marcare i pacchetti di sua competenza; ad esempio, nella figura l'azienda ENI ha marcato i suoi pacchetti in rosso, mentre la Fiat ha marcato i propri pacchetti in blu.

In una situazione del genere si hanno una serie di problemi:

- Come si può essere sicuri che dei pacchetti marcati in un determinato modo non siano stati manomessi? Un pacchetto "rosso" è davvero di quel colore o è stato modificato?
- Come proteggersi nei confronti di chi gestisce la rete pubblica?

Esistono diverse tecniche per la realizzazione di una VPN:

- Utilizzo di indirizzamenti nascosti.
- Mediante routing protetto (tunnel IP).
- Uso di routing protetto e sicuro (tunnel IP sicuro).

5.8.1 VPN mediante rete nascosta

Questo tipo di VPN è basato sull'utilizzo di un indirizzamento non standard in modo da non essere raggiungibili da altre reti. Esempio di questa tecnica è l'uso di reti nascoste IANA secondo l'RFC-1918.

Da un punto di vista della sicurezza questo sistema è estremamente debole perché:

- Se qualcuno scopre gli indirizzi usati e decide di cambiarsi l'indirizzo e di entrare in un'altra classe esso sarebbe in grado di avere accesso in un'altra rete.
- Chi ha il controllo dell'infrastruttura può leggere e manipolare i pacchetti in transito.

5.8.2 VPN mediante tunnel

Con questa tecnica i router provvedono ad incapsulare i pacchetti di rete all'interno di altri pacchetti, adottando degli instradamenti in modo che le diversi sedi di un'azienda potranno solamente comunicare tra di loro. Le tecniche di incapsulamento possono essere diverse:

- Se si vuole rimanere a livello IP è possibile effettuare un incapsulamento IP in IP.
- IP over MPLS.
- Altre implementazioni.

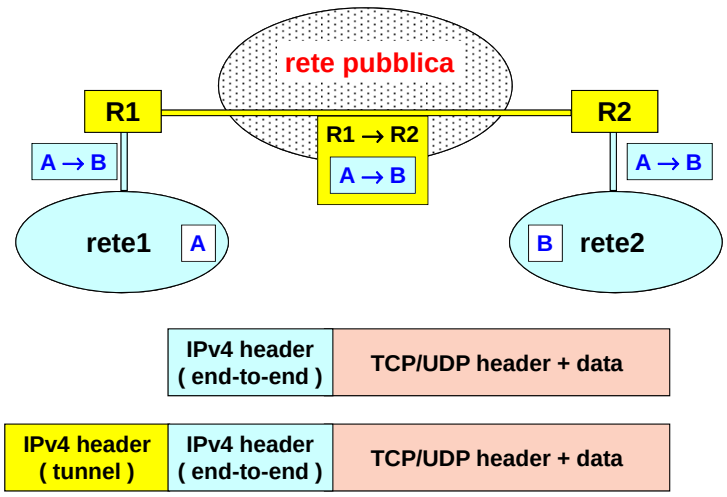
La cosa importante comunque a prescindere dal tipo di implementazione che si sceglie è che non bisogna usare per l'instradamento gli indirizzi originali ma si deve creare un nuovo indirizzamento che vincoli i pacchetti ad essere scambiati solo tra le sedi appartenenti alla stessa VPN logica.

Anche in questo caso i router controllano l'accesso alle reti mediante delle ACL (Access Control List).

Questa tecnica serve principalmente al fornitore dei servizi di telecomunicazione a proteggersi contro gli utenti che cercano di avere accesso a reti diverse dallo loro rete di appartenenza. Si ha quindi un minimo di sicurezza nel caso in cui un utente di una classe cerca di entrare in un'altra classe.

Tuttavia non si ha nessuna protezione per gli utenti del servizio di VPN nei confronti di chi gestisce la rete. Questo perché chi ha il controllo della rete potrà sempre andare a manipolare tutti i pacchetti a suo piacimento.

VPN mediante tunnel IP



Nel grafico si possono vedere due reti IP (colore azzurro), un nodo A che desidera comunicare con un nodo B e in mezzo ai due nodi una rete pubblica.

Il pacchetto originale mandato dalla rete1 ha un header end-to-end nel quale il mittente risulta essere il nodo A e il destinatario invece è il nodo B; tale header è seguito dal corrispondente payload.

Nel momento in cui questo pacchetto arriva al border router R1 della rete pubblica quest'ultimo vedrà che il pacchetto ha come destinazione la rete gestita dal router R2. Come conseguenza il pacchetto verrà incapsulato in un nuovo pacchetto, in cui l'header di livello IP avrà come mittente il router R1 e come destinatario il router R2. In altre parole il pacchetto originale diventa il payload di un nuovo pacchetto, al quale gli viene aggiunto un nuovo header che mette in collegamento gli end-to-end point del tunnel.

Come conseguenza di questa operazione il pacchetto incapsulato viaggia come in una galleria dentro la rete pubblica, e una volta giunto al router R2 questo andrà a rimuovere l'header del tunnel andando a ripristinare il pacchetto originale, e infine andrà a consegnarlo al nodo di destinazione.

Com'è possibile osservare da questo esempio chi gestisce la rete pubblica è libero di manipolare i dati a suo piacimento, mentre invece i gestori delle reti 1 e 2 non potranno cercare di avere accesso in altre reti perché saranno i border router a forzare l'instradamento.

Tunnel IP: frammentazione

Realizzare dei tunnel IP ha un costo, nel senso che esso si basa sull'incapsulare un pacchetto IP dentro un altro pacchetto e questo può portare a generare un pacchetto con una dimensione superiore all'MTU. Se questo si verifica bisogna ricorrere alla frammentazione, che porta ad avere una perdita di prestazioni massima del 50%.

Il calo di prestazioni reale comunque dipende dalla dimensione dei pacchetti. In generale le applicazioni che soffrono maggiormente sono le applicazioni di trasferimento bulk, ossia quelle non interattive.

5.8.3 VPN mediante tunnel IP sicuro

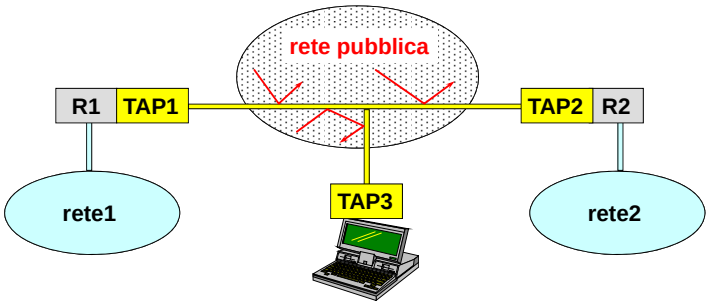
Questo tipo di VPN si basa sempre sull'utilizzo di un tunnel, ma i pacchetti prima di essere trasferiti vengono protetti. Questo viene fatto principalmente per proteggersi nei confronti del gestore della rete pubblica. I pacchetti possono essere protetti con:

- Digest, in modo da proteggere integrità e autenticazione. Questo evita che il gestore della rete possa modificare dei pacchetti o addirittura aggiungerne.
- Cifratura per la segretezza, in modo che non sia possibile leggere i pacchetti.
- Numerazione per evitare attacchi di tipo replay.

La cosa importante di questa tecnica è che la protezione deve essere fatta direttamente dal cliente della VPN, e non dal gestore. Se inoltre il cliente decide di utilizzare degli algoritmi crittografici forti, allora l'unico attacco possibile è impedire le comunicazioni mediante DoS.

Questa tecnica di VPN è anche definita S-VPN, ossia Secure VPN.

Esempio



In questo esempio oltre ad esserci i router R1 e R2 che si occupano del processo di incapsulamento si ha un altro nodo nel quale vengono implementate le funzionalità di sicurezza aggiuntive che il cliente desidera avere.

Le frecce all'interno della rete pubblica stanno ad indicare che eventuali attacchi "rimbalzano", non hanno alcun effetto.

5.9 IPsec

IPsec è l'architettura proposta dall'IETF per fare sicurezza al livello 3 sia per IPv4 sia per IPv6. IPsec permette di:

- Creare VPN su reti non fidate.
- Fare sicurezza end-to-end tra due nodi di una rete IP.

Per implementare le sue funzionalità IPsec modifica il protocollo IP andando ad aggiungere due header particolari:

- *Authentication Header (AH)* che fornisce come funzionalità aggiuntive integrità (del pacchetto), autenticazione (del mittente) e no replay.
- *Encapsulating Security Payload (ESP)* che fornisce tutte le funzionalità dell'header AH più la funzionalità di avere riservatezza.

Osservando le funzionalità offerte si può intuire come esse siano basate sull'uso di chiavi crittografiche che devono essere negoziate tra i nodi della rete IP. Si utilizza a questo proposito un protocollo per scambio delle chiavi chiamato *Internet Key Exchange (IKE)*, che serve a negoziare gli algoritmi e le chiavi da usarsi per implementare le funzionalità di sicurezza di AH e ESP.

Riassumendo IPsec fornisce autenticazione dei pacchetti IP:

- Integrità dei dati trasportati.
- Identificazione del mittente.
- Protezione (parziale) da attacchi di tipo replay.

La riservatezza dei pacchetti IP viene invece offerta tramite la cifratura dei dati.

5.9.1 IPsec Security Association (SA)

Tutte le funzionalità di IPsec vengono realizzate mediante il concetto di Security Association, ossia due nodi di una rete IP per poter beneficiare del protocollo IPsec devono prima negoziare fra di loro una associazione di sicurezza (qualche cosa che decida quali caratteristiche di sicurezza deve avere il traffico tra loro due).

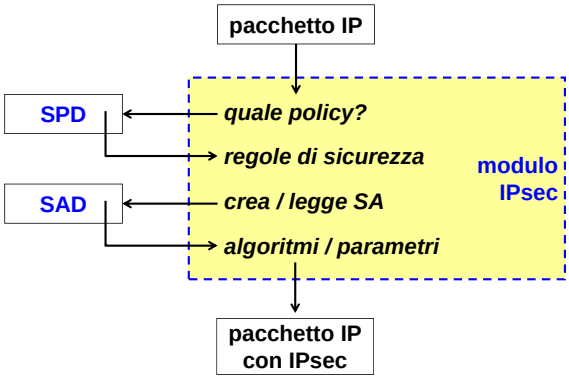
Una SA è un'associazione unidirezionale, e di conseguenza se si vuole proteggere il traffico in entrambe le direzioni tra due nodi si dovrà andare a creare due SA.

Non si ha nessun obbligo di dare le stesse caratteristiche di sicurezza ad entrambe le SA.

Per gestire le SA e in generale il traffico di tipo IPsec, ogni nodo che implementa IPsec deve anche implementare due DB (DB dal punto di vista logico):

- *Security Policy Database (SDP)* il quale contiene le security policy da applicare ai diversi tipi di comunicazione. Esso può essere configurato a priori (es. manualmente) oppure può anche essere agganciato a dei sistemi automatici (es. ISPS - Internet Security Policy System).
  La policy fornisce informazioni su che caratteristiche di sicurezza deve avere un determinato tipo di traffico.
- *SA Database (SAD)* che contiene l'elenco delle SA attive e delle loro caratteristiche (algoritmi, chiavi, parametri).

5.9.2 Come funziona IPsec (spedizione)



Si supponga di osservare un pacchetto IP outgoing da un nodo di rete.

Se è stata attivata la protezione IPsec per il traffico IP allora il pacchetto IP quando esce dal livello 3 dello stack OSI, prima di essere inviato al livello 2 viene intercettato dal modulo di IPsec. Il pacchetto viene ispezionato e il modulo, dopo aver effettuato un lookup all'interno dell'SPD, determina in base alle caratteristiche del pacchetto se è necessario applicare o meno una qualche policy di sicurezza.

Se il pacchetto non necessita di alcuna policy allora esso passerà direttamente al livello 2 dello stack, mentre in caso positivo il modulo andrà ad assegnare una specifica SA al pacchetto. Se questo è il primo pacchetto che viene scambiato fra i due nodi di rete allora sarà necessario creare una SA e registrarla all'interno del SAD; se invece questo non è il primo pacchetto fra il flusso dei due nodi, ma è un pacchetto che corrisponde ad una SA già registrata allora sarà necessario fare un lookup dentro il SAD in modo da recuperare i dati associati a tale SA, e andare così ad applicarli al pacchetto.

Il risultato di queste operazioni sarà un pacchetto IP aumentato con IPsec. Il livello 2 avrà poi il compito di trasmetterlo a destinazione.

5.9.3 Dove si applicano le funzionalità di IPsec?

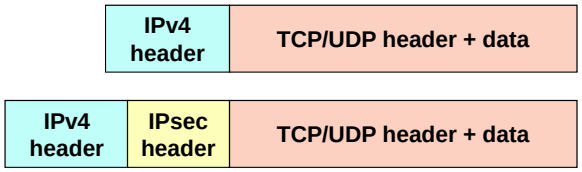
Nel momento in cui si va a proteggere un pacchetto bisogna prestare molta attenzione su quali componenti del pacchetto possono essere protetti.

Per quanto riguarda la funzionalità di riservatezza tipicamente la si applica solamente al payload perché se si andasse a cifrare anche l'header IP i router intermedi non avrebbero a disposizione alcune informazioni necessarie per le decisioni di instradamento.

Riguardo invece l'autenticazione e integrità ad una prima analisi si potrebbe pensare che essa sia applicata all'intero pacchetto. Tuttavia questo non è vero, in quanto alcuni campi all'interno dell'header vengono modificati a mano a mano che il pacchetto attraversa i vari router di comunicazione. Come conseguenza queste due funzionalità di sicurezza sicuramente andranno a coprire il payload, ma anche i campi fissi all'interno dell'header IP.

5.9.4 IPsec in transport mode

Questa modalità di utilizzo di IPsec consiste nel prendere il pacchetto originale e inserire tra il l'header IP e il payload l'header IPsec aggiuntivo.



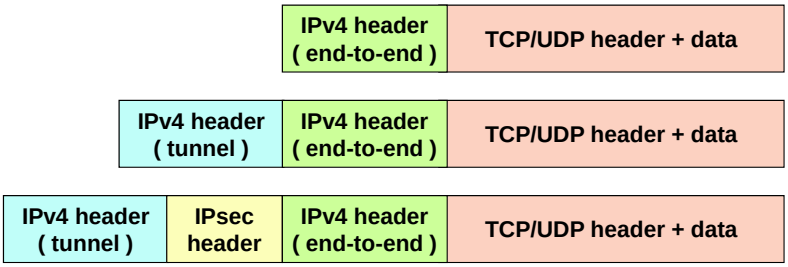
Effettuando tale operazione bisogna anche andare ad aggiornare il campo protocol all'interno dell'header IP, in modo che risulti che l'header IP è usato per trasportare un header IPsec. È come se IPsec venisse considerato come uno pseudo-protocollo di livello 4, ma in realtà all'interno dell'header IPsec si ha l'indicazione del protocollo di livello 4 effettivamente utilizzato.

IPsec in transport mode viene solitamente usato per fare sicurezza end-to-end, ossia usato dagli host e non dai gateway. L'eccezione si ha quando i nodi intermedi si comportano loro stessi da end-node, ad esempio quando si utilizza SNMP o ICMP.

Il vantaggio di questa modalità è che le operazioni da eseguire sono poche, e quindi è computazionalmente leggero. Lo svantaggio è che non si fornisce nessuna protezione dei campi variabili dell'header IP.

5.9.5 IPsec in tunnel mode

Questa modalità di utilizzo è quella che permette di realizzare le VPN, e quindi viene solitamente utilizzata dai gateway.



L'idea di base è quella di prendere il pacchetto originale (header+payload), andare ad incapsularlo dentro un altro pacchetto IP e al pacchetto risultante si va ad applicare la protezione IPsec. Con questa tecnica l'header IPsec va a proteggere tutto il contenuto del pacchetto originale, incluso il suo header; le uniche parti che non sono protette sono le parti variabili del tunnel.

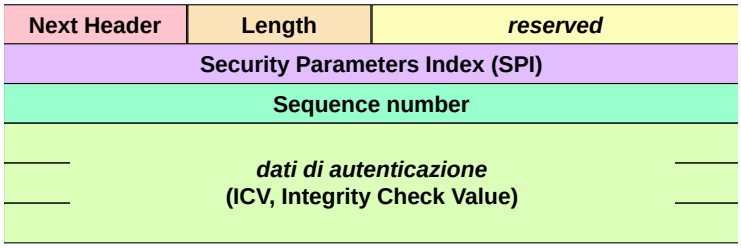
Il vantaggio di questa tecnica è la protezione del pacchetto originale, inclusi i sui campi variabili (che non sono più variabili, rimangono fissi all'interno del tunnel). Lo svantaggio è che si ha un costo computazionale maggiore rispetto ai pacchetti in transport mode, perché bisogna prima creare un tunnel e poi andare a proteggerlo.

5.9.6 Authentication Header (AH)

Sono state implementate due versioni di questo header:

- La prima versione (RFC-1826) permetteva di avere solamente integrità dei dati e autenticazione del mittente. Come keyed digest erano stati previsti gli algoritmi keyed-MD5 (obbligatorio) e il keyed-SHA-1 (opzionale).
- La seconda versione (RFC-2402), che è oggi quella più usata, oltre ad integrità e autenticazione ha aggiunto la funzionalità del no replay. Gli algoritmi utilizzati da questa versione sono l'HMAC-MD5-96 e l'HMAC-SHA-1-96.

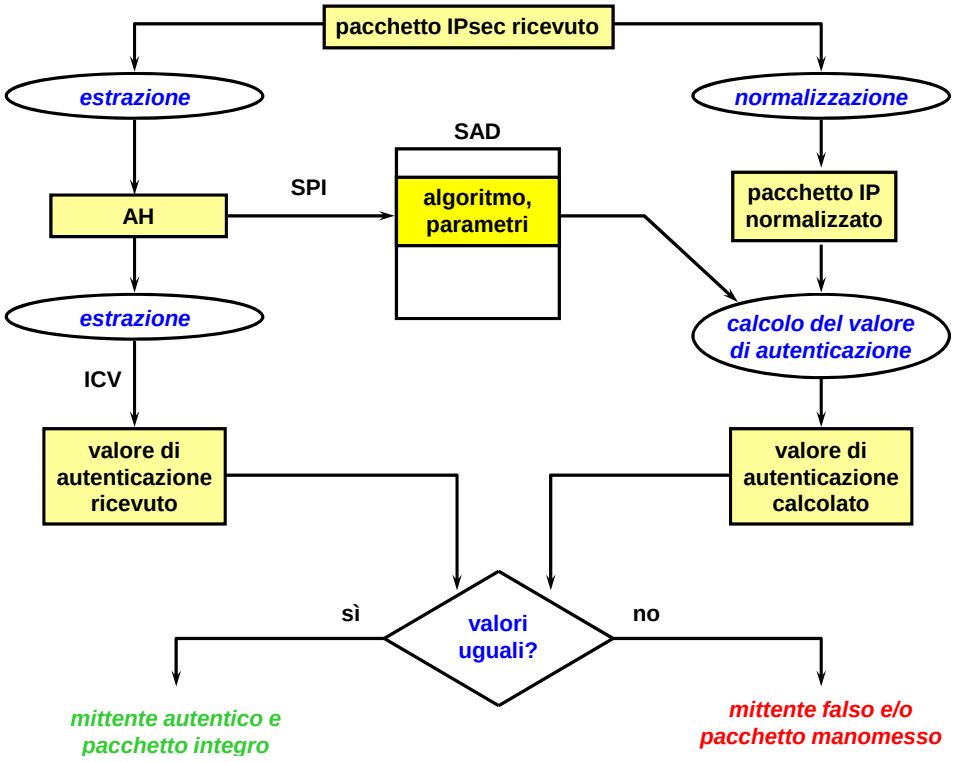
AH - Formato (RFC-2402)



Questo header risiede tra l'header IP e il payload, e contiene tutte le informazioni necessarie per autenticare e per assicurare l'integrità del pacchetto.

- *Next Header* (1 byte): conserva informazioni del protocollo di livello 4 originale.
- *Length* (1 byte): lunghezza.
- *Reserved* (2 byte): bit riservati per usi futuri.
- *Security Parameter Index, SPI* (4 byte): indice nella tabella SAD. Questo identificativo serve ad associare l'header ad una particolare SA. Per trovare quali sono gli algoritmi e in generale i parametri applicati al pacchetto contenente un header AH, il mittente e il ricevente devono effettuare un lookup nelle loro rispettive tabelle di SAD usando una tripletta: indirizzo_mittente/indirizzo_destinatario/SPI.
- *Sequence Number* (16 byte): campo utilizzato per numerare i pacchetti IP appartenenti alla singola SA. Serve ad evitare attacchi di tipo replay.
- *Dati di autenticazione* (12 byte): questi bit costituiscono il cosiddetto ICV (Integrity Check Value). Dati necessari a fornire autenticazione ed integrità. È il mittente a calcolare questi dati; il destinatario ha il compito di ricalcolarli sul pacchetto ricevuto e di andare a confrontarli.

AH - Procedura



Si riceve un pacchetto IPsec, protetto con AH, e si vuole sapere se esso sia integro ed autentico.

La prima operazione che viene eseguita è quella di andare ad estrarre dal pacchetto i dati relativi ad AH. Da questi dati si può ricavare l'ICV, e in questo modo è possibile sapere il valore di autenticazione che il mittente ha calcolato ed inserito all'interno del pacchetto.

Fatta questa prima operazione bisogna controllare se questo valore di autenticazione corrisponde ai dati ricevuti. Si effettua quindi una normalizzazione del pacchetto IPsec (in breve ci si mette nelle stesse condizioni in cui il calcolo è stato fatto dal mittente). Una volta che si ha a disposizione il pacchetto IP normalizzato occorre calcolare il keyed-digest; per fare questo però è necessario sapere l'algoritmo e la chiave, i quali sono ottenibili effettuando un lookup all'interno della SAD basato sull'SPI contenuto all'interno dell'AH.

Si è ora in grado di calcolare il valore di autenticazione sui dati ricevuti. Se il valore di autenticazione ricevuto risulta essere uguale a quello calcolato allora il mittente è autentico e il pacchetto è integro, mentre in caso contrario il mittente è falso e/o il pacchetto è stato manomesso. In quest'ultimo caso il pacchetto dovrà essere scartato.

In caso l'operazione avesse risultato positivo, chi è stato autenticato? L'unica certezza che si ha è che il pacchetto è stato spedito da colui che ha negoziato con il destinatario l'SA. Chi sia questo colui dipende dalla tipologia di autenticazione usata nel momento di creazione della SA.

AH - Normalizzazione

Per normalizzazione si intende mettersi nelle stesse condizioni del mittente, ossia viene azzerato il campo TTL (il mittente metterebbe un certo valore, mentre il destinatario lo riceverebbe pari al valore iniziale meno il numero di router che sono stati attraversati). Questa operazione viene effettua sia dal mittente che dal destinatario, ma solo ai fini del calcolo del valore di autenticazione.

Nel caso in cui il pacchetto contenesse un Routing Header allora occorre:

- Fissare il campo destinazione all'indirizzo del destinatario finale.

- Fissare il contenuto del routing header al valore che avrà a destinazione.
- Fissare il campo Address Index al valore che avrà a destinazione.

Qualunque altra opzione che cambi durante il transito (ossia che presentino il bit C, Change En Route) in rete devono anch'esse essere azzerate.

AH - HMAC-SHA1-96

Dato un pacchetto M si effettua la normalizzazione generando il pacchetto M'. Il funzionamento di questo algoritmo è quindi il seguente:

- Si effettua un allineamento a 160 bit di M' (aggiungendo bit a zero), andando così a generare $M'p$.
- La chiave K viene anch'essa allineata a 160 bit (aggiungendo bit a zero), in modo da creare la chiave K_p.
- Dati i valori fissi ip = 00110110 e op = 01011010 (entrambi ripetuti in modo da formare 160 bit) si può a questo punto calcolare la base di autenticazione seguendo la formula

$$B = \text{sha1}\left((K_p \oplus \text{op}) \,||\, \text{sha1}\left((K_p \oplus \text{ip}) \,||\, M'p\right)\right)$$

- Poiché la dimensione di B è variabile a seconda dell'algoritmo usato, il che andrebbe a generare pacchetti IPsec di dimensioni non fisse, allora si ha che

$$\text{ICV} = 96 \text{ leftmost bit di } b$$

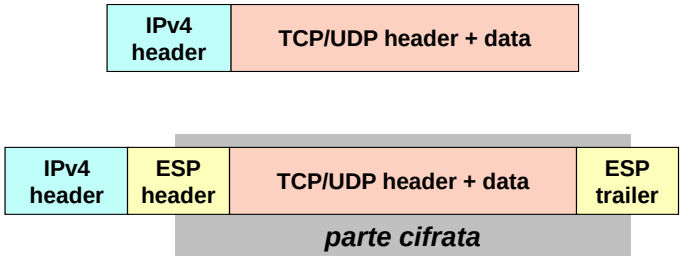
5.9.7 Encapsulating Security Payload (ESP)

Nella sua prima versione (RFC-1827) i pacchetti ESP erano ortogonali ai pacchetti AH, ossia fornivano solo riservatezza. Se quindi si desidera avere autenticazione, integrità, no replay e riservatezza su uno stesso pacchetto era necessario andare ad implementare sia AH sia ESP.

Nella versione successiva (RFC-2406) si è deciso di implementare dentro ESP anche la maggior parte delle funzionalità di AH, in particolare la parte di autenticazione, integrità e no replay. Si aveva solo un'unica eccezione: il pacchetto ESP non include l'header IP, e quindi le funzionalità di sicurezza riguardano solo il payload di un pacchetto. Il vantaggio è la riduzione della dimensione del pacchetto.

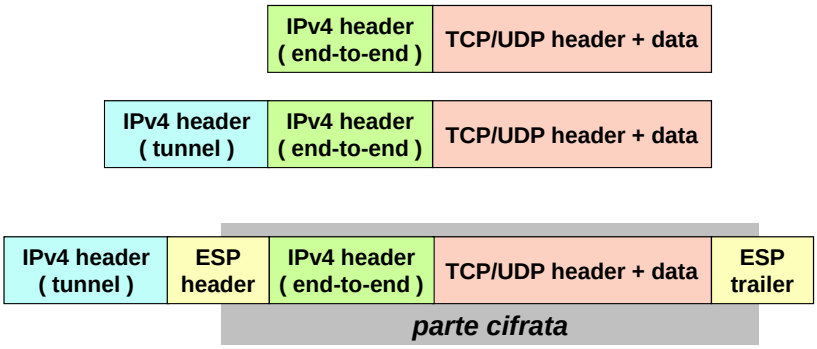
ESP in transport mode

ESP in transport mode la parte cifrata comprende solamente il payload; l'header originale rimane completamente in chiaro.



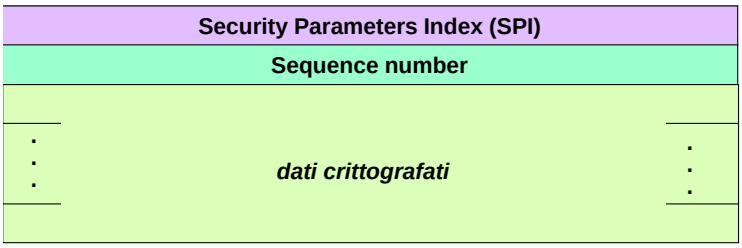
ESP in tunnel mode

Se ESP viene invece applicato in tunnel mode, visto che il payload viene ad essere tutto il pacchetto originale vengono di conseguenza nascosti anche i veri mittente e destinatario del pacchetto.



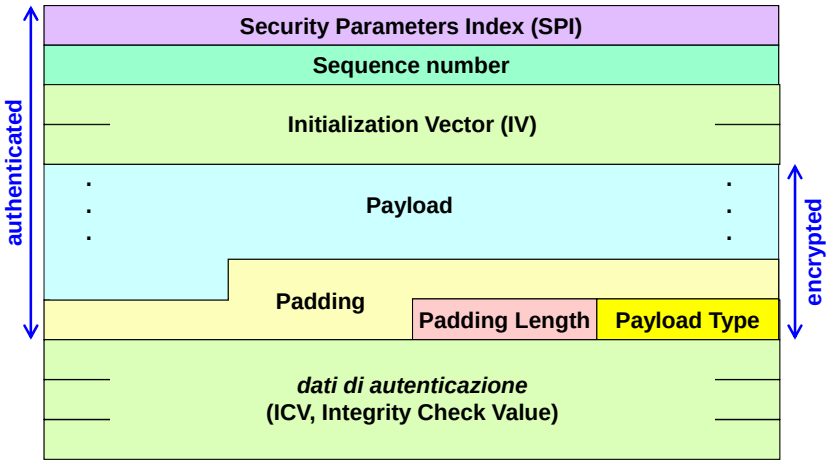
Quello che rimane in chiaro è l'header di tunnel. Un eventuale attante quindi non sarà in grado di sapere chi sta utilizzando il tunnel per comunicare.

ESP - Formato (RFC_2406)



- *Security Parameter Index* (SPI): indicazione di quale SA fa riferimento questo pacchetto.
- *Sequence Number*.
- *Dati crittografati*: per poter leggere questi dati è necessario essere a conoscenza della SA usata per creali.

Supponendo che i dati siano stati crittografati usando l'algoritmo DES-CBC, il formato del pacchetto è il seguente:



Dopo il Sequence Number ci sono i 128 bit del vettore di inizializzazione (*Inizialization Vector, IV*), campo usato dall'algoritmo CBC; a questo campo segue il *payload*. Poiché il DES richiede che il payload sia un multiplo di 64 bit, esso sarà seguito da del *Padding*: la dimensione del padding aggiunto sarà indicata nel campo *Padding Length*, il quale sarà seguito dal campo *Payload Type* ossia qual è il protocollo di livello superiore che viene trasportato.

I campi payload, Padding Length e Payload Type sono i dati che vengono cifrati, mentre i dati autenticati sono questi ultimi campi in aggiunta al vettore di inizializzazione, al Sequence Number e al SPI.

I *dati di autenticazione* (96 bit) sono posti in fondo al pacchetto. L'ICV è stato calcolato esattamente nello stesso modo in cui veniva calcolato per AH, con l'eccezione che non dovendo andare a trattare l'header IP non è necessaria la procedura di normalizzazione poiché in questo caso i dati sono tutti fissi.

5.9.8 Dettagli implementativi

Poiché è possibile andare a scegliere tra molti algoritmi, sono state definite due cripto-suite (insieme di algoritmi) nell'RFC-4308 in modo da facilitare l'interoperabilità nel momento in cui si realizzano delle VPN:

- VPN-A: prevede l'utilizzo di pacchetti ESP con cifratura 3DES-CBC, e con autenticazione ed integrità data da HMAC-SHA1-96. Questo rappresenta il livello minimo di sicurezza a cui è possibile fare affidamento.
- VPN-B: si basa sempre sull'utilizzo di pacchetti ESP ma con cifratura AES-128-CBC, e la parte di autenticazione data da AES-XCBC_MAC-96. Questa soluzione, oltre ad offrire un livello di sicurezza maggiore rispetto alla soluzione precedente, ha il vantaggio di usare ES sia per la parte di integrità sia per la parte di autenticazione, e quindi non è necessario andare ad implementare ulteriori algoritmi.

Molto usati sono anche gli algoritmi NULL, sia per la parte di autenticazione che per quella di crittografia. Questo permette di usare un unico formato (ad esempio ESP) e di modulare ciò che si vuole andare ad implementare. Se ad esempio si volesse avere ESP solo per riservatezza, è possibile andare a specificare NULL per la parte di keyed-digest e di conseguenza si avrà solo riservatezza.

In generale quindi questi algoritmi servono ad avere un trade-off tra protezione e prestazioni.

L'ultimo dettaglio implementativo ma molto importante è quello che riguarda il Sequence Number. Esso offre una protezione da replay, ma solamente in modo parziale; questo perché esso è associato ad una finestra che riguarda quali pacchetti sono coperti dalla protezione da replay. Lo standard richiede come minimo che vengano considerati gli ultimi 32 pacchetti ricevuti, ma il consiglio è quello di considerare gli ultimi 64 per avere una migliore protezione.

5.9.9 Protezione da replay in IPsec

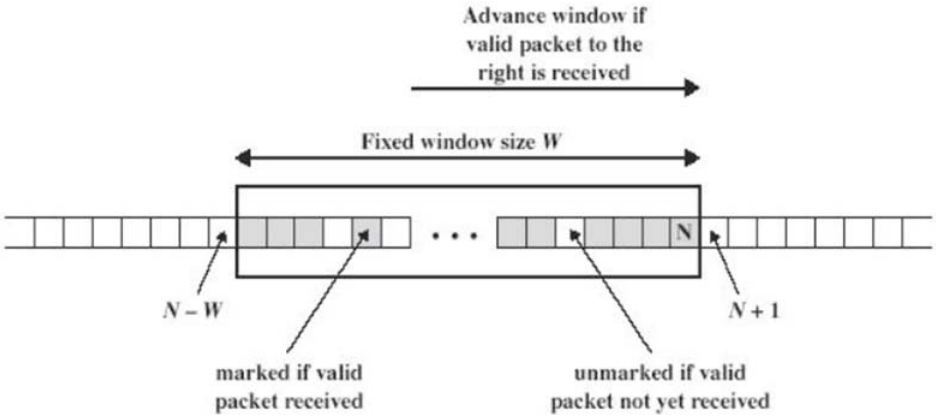
Innanzitutto è bene sottolineare che IPsec si applica a pacchetti IP. La rete IP tuttavia non è affidabile e quindi i pacchetti potrebbero andare persi; di conseguenza IPsec non può fornire in alcun modo protezione su packet filtering e packet cancellation. La protezione da replay è solamente parziale perché IP ha il problema di essere un protocollo non sequenziale, ossia l'ordine con cui i pacchetti vengono ricevuti può non corrispondere con l'ordine con cui i pacchetti sono stati trasmessi. Questo significa che nel momento in cui si va a verificare un attacco replay non è possibile andare a vedere solamente il numero di sequenza, in quanto un pacchetto con un numero di sequenza minore potrebbe arrivare a destinazione dopo un pacchetto con numero di sequenza maggiore. Per sapere se un pacchetto è già stato ricevuto o meno quindi non basta tenere il numero di sequenza del pacchetto con numero più elevato ricevuto, ma è necessario tenere traccia di tutti i pacchetti che si sono ricevuti. Ma questo è impossibile, perché sarebbe necessario avere una memoria enorme per tenere traccia di tutti i pacchetti. Da qui nasce il

concetto di finestra, ossia si ha un buffer in cui si tiene traccia degli ultimi numeri di sequenza degli ultimi N pacchetti ricevuti.

Questo concetto è implementato lato mittente nel modo seguente:

- Quando si crea una SA il mittente inizializza il Sequence Number a 0.
- Quando si invia un pacchetto si incrementa il Sequence Number.
- Quando si raggiunge il Sequence Number massimo si negozia una nuova SA (altrimenti si avrebbe overflow con conseguente attacco replay automatico).

Il destinatario invece utilizza una finestra di dimensione fissa W.



I pacchetti in grigio sono quelli che sono già stati ricevuti, mentre in bianco sono i pacchetti che devono ancora essere ricevuti.

Quando si riceve un pacchetto alla destra della finestra, quest'ultima si sposta a destra fino ad includerlo. In questo modo vengono esclusi dei pacchetti più vecchi.

Se avviene il replay di un pacchetto che sta dentro la finestra, esso viene identificato subito. Se invece il replay ha come oggetto un pacchetto all'esterno della finestra, non si ha modo di sapere se si tratta di replay o meno. In quest'ultimo caso è possibile avere due strategie:

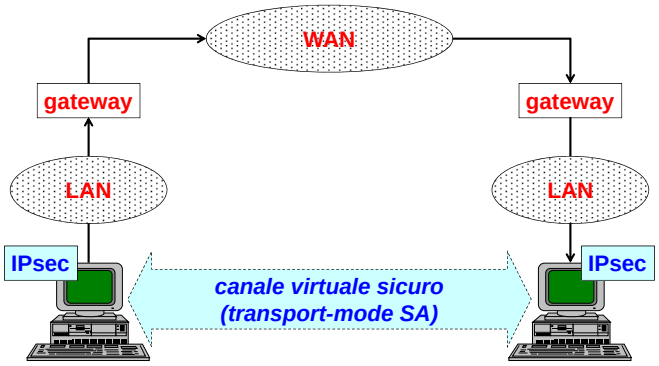
- Il pacchetto viene accettato comunque, e viene mandato ai livelli superiori. Se il livello superiore è TCP allora sarà quest'ultimo a gestire il pacchetto nel modo più opportuno (andando ad accettarlo nel caso in cui fosse realmente un pacchetto mancante, oppure andando a rifiutarlo se esso è un pacchetto che è già stato ricevuto), ma nel caso in cui si avesse UDP la situazione è meno gestibile in quanto UDP non ha il concetto di segmento e quindi il pacchetto potrebbe essere accettato anche se esso è già stato ricevuto in precedenza. Gli attacchi UDP quindi sono molto più pericolosi per UDP rispetto a TCP.
- Se il pacchetto è troppo vecchio, e quindi fuori dalla finestra, lo stack IPsec lo va a scartare in ogni caso. Questa soluzione garantisce una migliore difesa da replay, ma potrebbe causare dei rallentamenti in quanto potrebbe causare delle ritrasmissioni.

Riassumendo la protezione da attacchi replay dipende dalla dimensione della finestra e dalla strategia usata per la gestione di pacchetti ricevuti al di fuori della finestra.

5.9.10 Architetture di protezione

Affinché un sistema possa essere dichiarato IPsec-compliant deve poter implementare almeno quattro architetture base.

End-to-end security



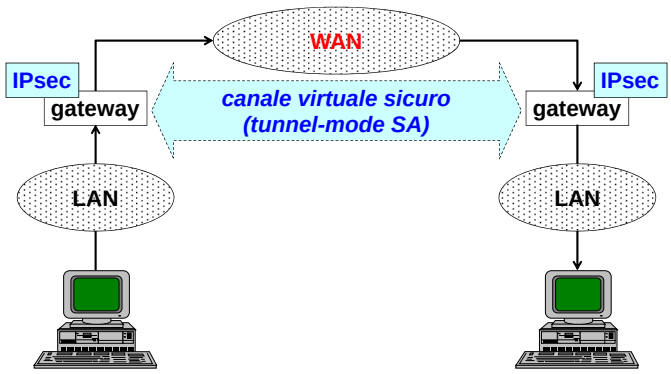
Questa architettura prende questo nome perché è quella in cui IPsec viene usato per proteggere il traffico direttamente fra due nodi client-server.

Questo significa che i due nodi su cui è stato installato il software IPsec negoziano tra di loro una SA in modalità transport mode. In altre parole i due nodi si creano un canale virtuale sicuro, con le caratteristiche di sicurezza che hanno negoziato, che rende la protezione del traffico da essi generato indipendente da tutto: non è importante se le LAN non sono sicure, se i gateway sono gestiti da persone non fidate o chi implementa o gestisce le WAN. Se le due macchine sono sicure allora si avrà la protezione completa del traffico di rete, ad eccezione del DoS.

Questa architettura presenta sia dei vantaggi che dei svantaggi:

+ I due nodi sono completamente resi indipendenti e protetti da tutto il resto della rete, sia locale che geografica.
− È necessario andare ad installare IPsec su tutti i nodi della rete che si intende andare a proteggere; di conseguenza questa soluzione è usata principalmente quando si hanno pochi nodi da proteggere.
− Il fatto di andare ad installare IPsec sul singolo nodo presuppone che quel nodo abbia le capacità crittografiche necessarie; questo è molto importante per quanto riguarda il client, ma anche per quanto riguarda il server nel caso in cui il numero dei nodi fosse elevato.
− Nel caso che nel canale sia stato implementato riservatezza il gestore della rete locale non può andare a fare monitoraggio del traffico.

Basic VPN



Questa architettura implementa l'idea opposta rispetto all'architettura precedente: invece di andare ad implementare IPsec sugli end-node, si presuppone che le reti locali siano fidate e che

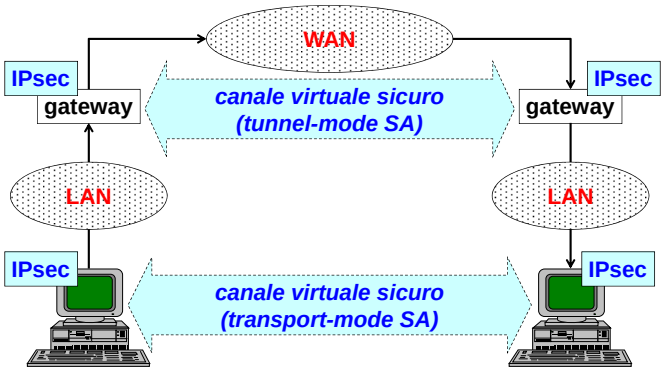
quelle pericolose siano solamente le reti geografiche. In base a questa idea IPsec viene installata sui gateway, ossia i punti di accesso fra due reti con livello di sicurezza differente: una rete sicura di cui ci si fida (nell'immagine è stata definita come LAN) e una di cui non si si fida (WAN).

Vantaggi e svantaggi:

+ Non occorre più andare a gestire un numero elevato di nodi, ma è sufficiente la gestione del solo gateway.
+ Se il gateway avesse dei problemi computazionali è possibile andare a prendere delle misure andando ad installare degli acceleratori crittografici.
+ La parte di rete fidata non essendo più cifrata può essere soggetta a monitoraggio.
− Non si ha più una protezione end-to-end. Questo significa che se c'è qualche malintenzionato all'interno della rete fidata questo è libero di eseguire qualunque tipo di attacco esso voglia.

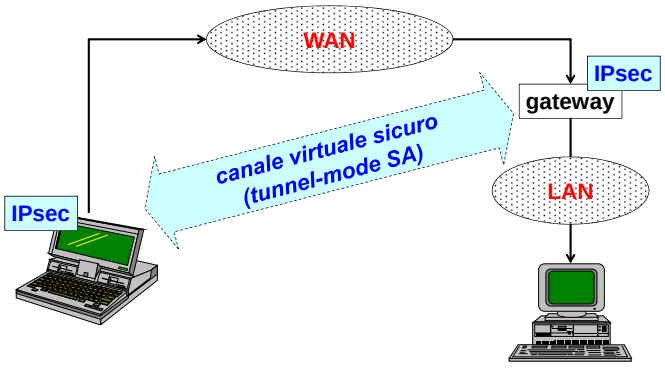
Oggi questa è una delle soluzioni più usate.

End-to-end security with basic VPN



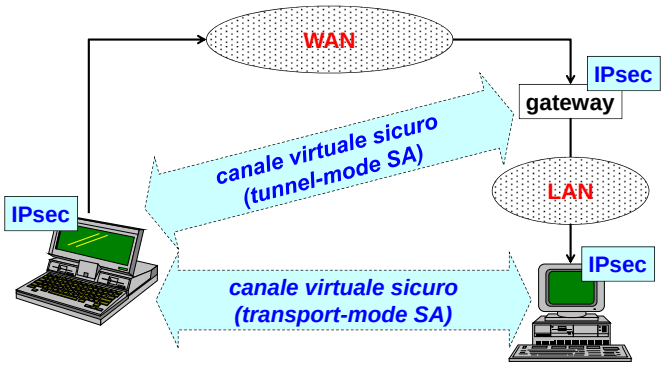
Questa architettura implementa una doppia linea di difesa, andando ad installare i moduli IPsec sia sugli end-node sia sui gateway. Questo può essere fatto sia per andare a raddoppiare il livello di difesa, sia per andare a dividere le difese: ad esempio sul canale virtuale sicuro transport-mode potrebbe essere implementato IPsec ma solo con AH, il quale garantirebbe autenticazione e integrità dei dati e permetterebbe al gestore della rete il monitoraggio del traffico, mentre la parte di riservatezza è implementata sui gateway nell'ipotesi che la parte più pericolosa per quanto riguarda lo sniffing sia la parte di rete non fidata.

Secure gateway



Oggi molta importanza sta acquisendo l'architettura secure gateway, in cui un nodo mobile viene dotato di software IPsec e va a realizzare un canale virtuale sicuro in tunnel-mode con un punto di accesso ad una rete. Con questa soluzione non solo si implementa riservatezza ma anche autenticazione (solitamente a livello applicativo, in modo da implementare un sistema di gestione degli accessi).

Secure remote access



In questa quinta architettura non soltanto si ha un canale virtuale sicuro in tunnel-mode verso il gateway, ma si ha anche un canale virtuale sicuro in transport mode verso il nodo finale. Questo viene fatto per raddoppiare le funzioni di sicurezza oppure per dividerle. in quest'ultimo caso tipicamente l'autenticazione è posta a livello del tunnel-mode in quanto si vuole tener traccia di chi sta cercando di accedere alla rete dall'esterno.

5.9.11 IPsec key management

Una componente fondamentale di IPsec è la parte di key management, che fornisce ai sistemi che comunicano chiavi simmetriche necessarie per l'autenticazione e/o la cifratura dei pacchetti. Il problema è andare a decidere come andare a distribuire queste chiavi; questo può essere fatto:

- Manualmente, andando a configurare le chiavi direttamente sui nodi che comunicano tra di loro.
- In modo automatico.

ISAKMP

Internet Security Association and Key Management Protocol è il protocollo usato per negoziare le chiavi e stabilire le SA.

Esso definisce tutte le procedure che servono per negoziare, stabilire, modificare e cancellare le SA. Questo protocollo è un framework, nel senso che esso viene usato per trasportare pacchetti ma non indica il metodo da usarsi per lo scambio delle chiavi. Attualmente con ISAKMP viene molto spesso usato il protocollo OAKLEY, che realizza lo scambio autenticato delle chiavi simmetriche tra sistemi IPsec.

IKE

Internet Key Exchange altro non è che l'accoppiata tra ISAKMP e OAKLEY. Esso è stato standardizzato in quanto è una soluzione molto utilizzata.

Protocollo particolarmente complesso in quanto richiede di creare una prima SA non per proteggere lo scambio tra due nodi ma per proteggere lo scambio ISAKMP. Con questa SA viene protetta la negoziazione delle SA successive richieste da IPsec. La stessa SA ISAKMP può essere usata più volte per negoziare altre SA IPsec, sempre tra gli stessi nodi.

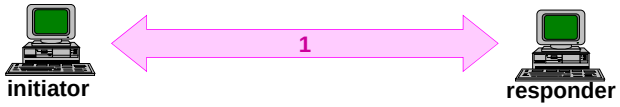


Figura 5.2: IKE phase 1 - negoziazione di una SA ISAKMP bidirezionale: "main mode" o "aggressive mode"

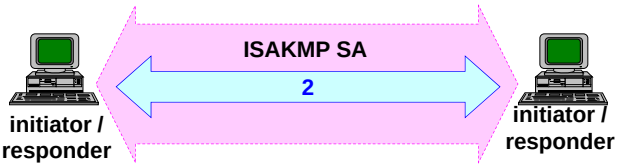


Figura 5.3: IKE phase 2 - negoziazione della SA IPsec: "quick mode"

Da un punto di vista funzionale ci sono due nodi che desiderano comunicare e uno dei due prende l'iniziativa; per tale ragione questo nodo è detto *initiator*. Questo nodo parla con un responder e va ad implementare la fase 1 di IKE, in cui viene negoziata una SA ISAKMP bidirezionale; questo può essere fatto in due modi, *main mode* o *aggressive mode*.

Creata questa prima SA, una volta che uno dei due nodi desidera aprire un canale di comunicazione verso l'altro nodo verrà creata una seconda SA ma ad uso di IPsec. Questa rappresenta la fase 2 di IKE che grazie al fatto che viene eseguita all'interno di un'altra SA può essere fatta in modo molto più veloce, e per tale ragione viene detta *quick mode*.

I diversi modi di funzionamento di IKE presentano le seguenti caratteristiche:

- *Main Mode*: è il più pesante di tutti perché richiede lo scambio di 6 messaggi crittografici. Il vantaggio è quello di proteggere l'identità delle parti.
- *Aggressive mode*: definito così perché richiede molti meno messaggi rispetto al modo precedente; in particolare sono sufficienti 3 pacchetti crittografici ha il vantaggio di essere estremamente più veloce ma non protegge l'identità delle parti.
- *Quick mode*: questa parte utilizza soltanto 3 messaggi, ed è usata solamente per la negoziazione della SA IPsec.
- *New Group Mode*: data una SA esistente, questa modalità serve solamente a cambiare i parametri crittografici. Si basa sull'uso di due messaggi.

Per quanto riguarda la parte di autenticazione, nel momento in cui si creano le SA è possibile fare:

- Digital Signature. Poiché si ha il supporto dei certificati X.509 è possibile avere non-repudiation della negoziazione IKE. In altre parole un utente non può negare di aver creato una certa SA.
- Public Key Encryption: metodo di autenticazione non basato sui certificati X.509 ma su metodi di cifratura a chiave pubblica, in modo da proteggere l'identità delle parti nell'aggressive mode.
- Revised Public Key Encryption: rispetto alla soluzione precedente questa risulta essere meno costosa perché utilizza solamente 2 operazioni a chiave pubblica.
- Pre-Shared Key: modo di autenticazione più semplice perché le chiavi non vengono negoziate, ma viene negoziato quali chiavi verranno utilizzate. Questo significa che i nodi avranno al loro interno delle chiavi, e durante lo scambio ISAKMP viene soltanto deciso quale chiave andare ad utilizzare. In questo caso il vincolo è che l'identificativo della controparte può essere solo il suo indirizzo IP; come conseguenza si ha un problema per gli utenti mobili, i quali non potranno usare le Pre-Shared Key in quanto il loro indirizzo IP è variabile.

5.9.12 IPsec - Requisiti di sistema

L'acquisto di un VPN concentrator non è indispensabile. È infatti possibile andare ad implementare IPsec su altri nodi, ad esempio:

- Router: è necessario disporre di un router con una CPU molto potente, oppure occorre andare a dotare il router di una acceleratore crittografico. In ogni caso è essenziale che il router non sia gestito in outsourcing, perché altrimenti si affiderebbe a terzi non soltanto la gestione della rete ma anche della sicurezza.
- Firewall: essendo anch'esso un punto di confine tra una rete sicura e non sicura è possibile andarvi ad installare il gateway IPsec. È anche in questo caso necessario avere una CPU molto potente.
- VPN concentrator: Per migliorare le prestazioni quando si utilizzano estensivamente la diverse modalità VPN molto spesso si comprano direttamente dei dispositivi hardware già dedicati a questi scopi. Queste VPN prendono il nome di VPN concentrator. Esse sono delle apparecchiature special-purpose che fungono da terminatori di tunnel IPsec, sia nel caso di accesso remoto di singoli client sia per creare delle VPN site-to-site.
  Il grosso vantaggio di queste macchine è che essendo macchine custom non soltanto hanno delle prestazioni molto elevate ma soprattutto hanno dei costi molto bassi. Inoltre esse offrono massima indipendenza dalle altre misure di sicurezza, permettendo quindi una segmentazione su più livelli delle misure di sicurezza.

5.9.13 Influenza di IPsec sulle prestazioni

IPsec ha un'influenza sulle prestazioni. In particolare implementare IPsec significa diminuire il throughput della rete; questo perché:

- Maggiore dimensione dei pacchetti: In trasport mode AH si vanno ad aggiungere come minimo 24 byte, mentre in trasport mode ESP-DES-CBC si ha un aumento minimo di 32 byte; in quest'ultimo caso l'aumento dipende da quanto padding viene aggiunto.
- Maggior numero di pacchetti: Bisogna infatti considerare i pacchetti necessari ad attivare la SA. Come conseguenza si avranno tempi di latenza superiore, nel senso che il tempo di set-up delle connessioni sarà più lento.

In generale la diminuzione del throughput è contenuta, ma questo non è sempre vero. Ad esempio l'implementazione di IPsec causa grandi rallentamenti nel caso in cui si va a sostituire un link punto-punto fisico con un link punto-punto virtuale; questo perché solitamente nel collegamento fisico si effettua una compressione a livello 2 dei dati, ma questa diventerebbe inutile o addirittura dannosa se associata a pacchetti ESP.

Per cercare di ovviare all'aumento di pacchetti è possibile comprimere i pacchetti prima di inviarli al modulo IPsec. Questo può essere fatto tramite il protocollo IPComp, che effettua una compressione del payload a livello IP, oppure la compressione può essere fatta direttamente a livello applicativo.

5.9.14 Applicabilità di IPsec

- Innanzitutto visto che IPsec richiede di negoziare una SA esplicita con un certo nodo, è possibile andare ad applicarlo solo ed esclusivamente a pacchetti unicast. IPsec non può quindi essere applicato a pacchetti broadcast, multicast o anycast. Più in generale IPsec non può essere applicato a tutti i pacchetti in cui non è possibile identificare con certezza il destinatario in quanto non sarà possibile avere una SA.
- I pacchetti unicast possono essere protetti solamente se è possibile identificare con certezza la controparte tramite una SA. Questo significa aver condiviso con la controparte una chiave

in modalità OOB, oppure la controparte è stata identificata tramite dei certificati X.509. Questo implica che in generale IPsec viene applicato a gruppi "chiusi": chiusi perché si ha avuto una distribuzione OOB delle chiavi, o chiusi virtualmente perché sono gruppi che si fidano delle medesime CA.

5.10 Sicurezza di IP

IPsec sicuramente protegge il traffico dei protocolli di livello superiore, ma non risolve completamente i problemi di sicurezza a livello IP.

IP in generale serve a trasportare non solo TCP e UDP ma anche molti altri protocolli. Purtroppo però IP risulta essere un protocollo molto poco sicuro perché:

- Gli indirizzi non sono autenticati.
- I pacchetti non sono protetti, e quindi non si ha integrità,autenticazione, riservatezza o replay.

Questo implica che sono attaccabili tutti i protocolli che usano IP come trasporto, soprattutto quelli di "servizio" ossia quelli di livello applicativo come ICMP, IGMP, DNS, RIP, ecc.

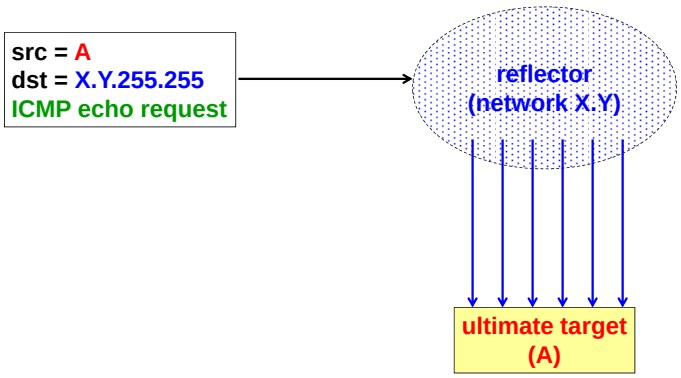
5.10.1 Sicurezza di ICMP

Il protocollo Internet Control and Management Protocol (ICMP) è vitale per la gestione della rete. Essendo privo di protezione è possibile avere moltissimi tipi di attacchi, in particolare:

- Un attaccante potrebbe sfruttare dei pacchetti destination unreachable (usati da un nodo intermedio per comunicare l'impossibilità di consegnare un pacchetto al destinatario) in modo da bloccare il traffico verso un nodo destinatario causando un DoS. Questo perché al ricevimento di un pacchetto destination unreachable il nodo mittente interrompe la trasmissione dei pacchetti.
- Tramite un pacchetto source quence (usato per rallentare la trasmissione dei pacchetti quando i buffer di un nodo intermedio sono saturi) un attaccante può andare a diminuire la velocità di trasmissione del mittente, e quindi generare un DoS.
- Possibilità di usare i pacchetti redirect (usati da un nodo intermedio per comunicare ad un altro nodo la strada migliore da seguire per consegnare dei pacchetti) in modo da generare un attacco Man-In-The-Middle logico.
- Uso di un pacchetto time exceeded for a datagram (usato quando si genera un loop nella rete) in modo da generare un DoS e far cadere la connessione tra due nodi.

Smurfing attack

Usando sempre ICMP è possibile fare una versione di ping flooding particolarmente devastante, che prende il nome di *Smurfing attack*.



Nello Smurfing l'attaccante invia un pacchetto ICMP echo request non ad un singolo nodo ma ad un'intera sotto rete; il mittente di questo pacchetto inoltre sarà la vittima (nell'esempio il nodo A). Tutti i nodi della sottorete che riceveranno questo messaggio risponderanno con un pacchetto ICMP echo respond, inondando la vittima di pacchetti.

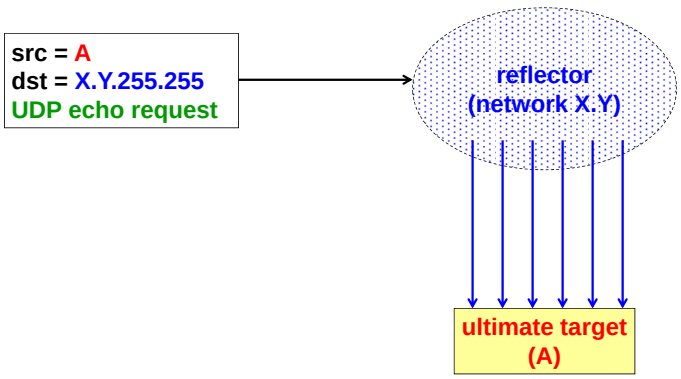
La sottorete oggetto del pacchetto dell'attaccante prende il nome di *reflector*, nel senso che rimbalza i messaggi ricevuti. L'efficacia di questo attacco è che l'attaccante invierà solamente un pacchetto, al ricevimento del quale ciascun nodo genererà una risposta che avrà come destinatario non l'attaccante ma la vittima.

Questo attacco non solo causa dei problemi alla vittima, ma causa dei problemi anche al gestore della rete in quanto l'enorme quantità di pacchetti ICMP potrebbe andare a saturare la banda o comunque generare dei grossi problemi di prestazioni.

In passato questo tipo di attacco è stato molto devastante. Con il tempo però si è capito che ricevere dei pacchetti broadcast dall'esterno della rete non ha molto senso, e quindi in generale i pacchetti IP broadcast vengono rifiutati se non sono generati da nodi interni alla rete. Se invece l'attaccante agisce direttamente dall'interno della sottorete l'unica soluzione è quella di usare degli strumenti di network management.

5.10.2 Fraggle attack

In seguito allo Smurfing attack si è passato al *Fraggle attack*. Questo non sfrutta il protocollo ICMP ma il protocollo UDP, e in particolare uno dei suo small services che permette di eseguire l'operazione di echo request/echo replay. Questo servizio mima la funzionalità fornita da ICMP, ma in realtà viene usata per testare le prestazioni e la funzionalità a livello 4.



Il funzionamento dell'attacco Fraggle è analogo a quello precedente, ma questa volta l'attaccante manda dei pacchetti in broadcast di tipo UDP echo request.

Questo attacco ha meno successo in quanto mentre ICMP è attivato di default su tutti i nodi di rete, gli UDP small services sono attivati a discrezione del sistemista.

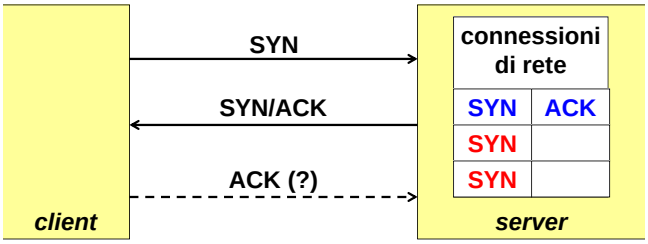
5.10.3 ARP poisoning

Questo attacco ha come oggetto il protocollo ARP, e sfrutta anch'esso il fatto che tale protocollo non è autenticato.

In particolare ARP poisoning viene usato per inserire dei dati falsi all'interno delle tabelle ARP, in modo da rendere possibili una serie di attacchi come ad esempio gli attacchi Man-In-The-Middle realizzati da Ettercap.

5.10.4 TCP SYN flooding

Il TCP SYN flloding è un attacco molto frequente e particolarmente fastidioso perché di tipo DoS, che mira a bloccare i server applicativi.



Per creare un canale TCP bisogna seguire la procedura three-way handshake:

- Un client manda un pacchetto di tipo SYN.
- Al ricevimento di tale pacchetto il server alloca una riga nella tabella delle connessioni e risponde con un pacchetto SYN/ACK.
- Il client a questo punto risponde con un ACK, in modo che il server possa andare a completare la riga precedentemente creata e quindi andare ad aprire la connessione.

In un attacco di TCP SYN flooding l'attaccante invia il SYN, ma non invierà mai l'ACK. In particolare l'attaccante continuerà a mandare pacchetti SYN, ai quali non seguirà mai nessun ACK.

L'effetto è che il server avrà moltissime righe nella tabella delle connessioni aperte solo parzialmente riempite. Questo causerà una saturazione della tabella sino a quando le richieste di connessione non andranno in timeout (valore tipico 75s). Se nel frattempo il server ricevesse un tentativo di connessione proveniente da un client, questo non sarebbe in grado di connettersi e quindi si avrebbe a tutti gli effetti un DoS.

Inoltre visto che l'attaccante non si vuole realmente connettere al server tipicamente i pacchetti SYN sono mandati con IP spoofing, ossia i pacchetti avranno un indirizzo IP falso. Come conseguenza non sarà per niente facile scoprire l'identità dell'attaccante.

Difesa da SYN flooding

Esistono alcune soluzioni per cercare di difendersi dall'attacco SYN flooding:

- Abbassare il timeout, ma questo potrebbe causare il rischio di eliminare client validi ma che sono lenti nel creare la connessione.
- Aumentare le dimensioni della tabella, possibile solamente se il server è effettivamente in grado di gestire un numero elevato di connessioni. Inoltre questa soluzione è aggirabile inviando più richieste.
- Usare un router come SYN interceptor, il quale serve a sostituire il server nella prima fase. Se l'handshake si completa con successo allora il router andrà a trasferire il canale al server. Si ha comunque in questo caso il rischio di saturare la tabella del router, problema risolvibile usando dei timeout aggressivi (con sempre il rischio di eliminare client validi).
- Usare un router come SYN monitor, con lo scopo di uccidere i collegamenti pendenti. Anche in questo caso se il timeout è troppo basso si ha il rischio di eliminare client validi.

SYN cookie

Questa idea è stata ideata da D.J.Bernstein, professore dell'Università di Chicago. Esso rappresenta l'unico sistema veramente efficace per evitare completamente il SYN flooding.

Il Professore ha osservato che il problema principale del SYN flooding sta nel fatto che i server tengono in memoria lo stato del collegamento con i vari client, e in presenza di troppe richieste di collegamenti si corre il rischio di saturazione.

L'idea è quindi quella di usare dei SYN cookie, ossia utilizzare il sequence number del pacchetto SYN-ACK per trasmettere un cookie al client e riconoscere così i client che hanno già inviato il SYN senza memorizzare niente nel server. Tale sequence number sarà calcolato in maniera crittografica, ossia un keyed-digest calcolato sulla data, ora e indirizzo IP del client che ha inviato il SYN.

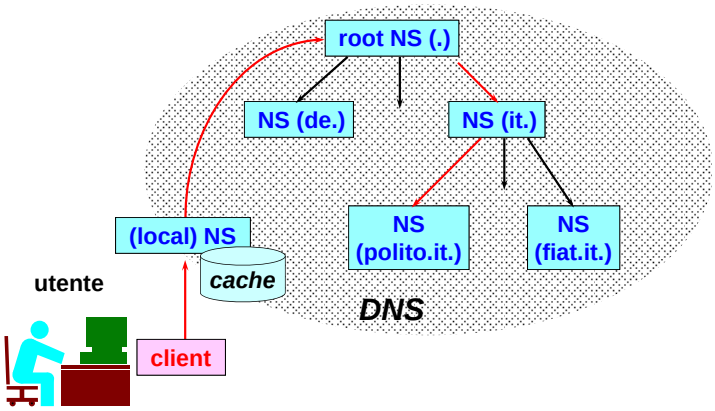
5.10.5 Sicurezza del DNS

Il protocollo DNS è quel protocollo che effettua una traduzione da nomi ad indirizzi IP e viceversa. La funzione svolta dal DNS è un servizio indispensabile.

È soggetto a parecchi tipi di attacco, in parte dovuti al fatto che per le query usa UDP in parte per l'uso di TCP per le cosiddette zone transfer, ossia per il trasferimento di grosse quantità di dati da un server primario a uno secondario.

DNS è stato sviluppato moltissimi anni fa, e per tale ragione è privo di qualunque forma di sicurezza. Negli ultimi anni si è cercato di sviluppare una versione sicura del DNS chiamata DNS-SEC.

Architettura del DNS



Normalmente il DNS viene usato da un utente non direttamente ma tramite un qualche programma, ossia un client DNS che sia in grado di "parlare" il protocollo DNS.

Il client deve essere configurato per conoscere quali sono i name-server (NS) di primo contatto, i cosiddetti local NS. Esso è definito local perché nel caso in cui l'utente fosse in una rete locale esso sarà anche il NS che conosce tutti i nodi presenti nella rete locale.

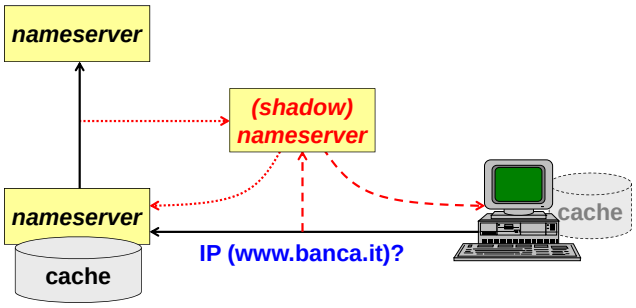
Nel caso in cui l'utente volesse avere accesso ad un nodo esterno alla propria rete locale, il local NS provvederà a contattare i cosiddetti root NS; questi NS sono quelli che gestiscono il dominio '.'. Il root NS che avrà ricevuto la richiesta dell'utente provvederà a contattare il NS di primo livello, il quale provvederà ad interrogare il NS di secondo livello e così via. Il processo continua fino a quando la richiesta dell'utente non verrà correttamente tradotta.

Una volta che l'utente riceve la risposta alla query essa verrà inserita in una cache, in modo da evitare una futura interrogazione. Queste cache tendono ad avere lunga durata, anche di una decina di giorni.

DNS shadow server

È uno degli attacchi più facili da fare.

Poiché i pacchetti non sono autenticati, se qualcuno fosse in grado di intercettare una DNS query sarebbe in grado di fornire una risposta errata.



Poiché le query sono basate su UDP il client riceverà anche la risposta vera dal NS, ma tale risposta verrà trattata come un pacchetto duplicato e quindi verrà scartata. Questo significa che verrà considerata la prima risposta che arriverà al client.

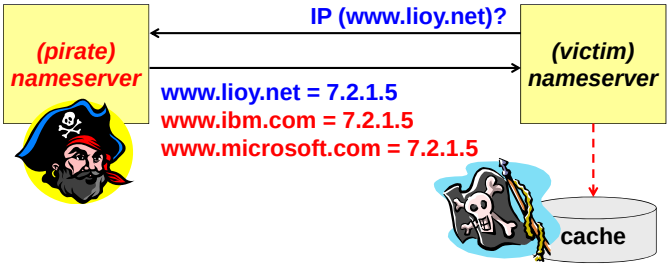
Occorre notare che se lo shadow NS risponde ad una query del client, le eventuali altre domande dello stesso tipo dovranno ricevere la stessa risposta. Se questo non avvenisse, e quindi il client ricevesse la risposta dal NS reale, il client andrebbe ad avere la traduzione corretta.

Per tale ragione è molto meglio fare lo sniffing non sulla tratta client-local NS ma sulla tratta geografica, ad esempio fra il local NS e il root NS. in questo modo se si va a fornire al local NS una risposta errata, questa verrà inserita nella cache e quindi verrà fornita a tutti i client della rete locale che faranno domanda.

È bene notare che le considerazioni precedenti si basano sul fatto che solamente i NS utilizzano una cache. Tuttavia Microsoft ha inserito sui suoi sistemi Windows una cache anche sui client; questo significa che gli attacchi fatti direttamente sulla rete locale sono più efficaci nei confronti dei client Windows perché è sufficiente fornire la risposta errata una sola volta. Sui client non Windows invece ci sarebbe bisogno sulla rete locale di continuamente fornire la risposta errata.

DNS cache poisoning

L'attacco precedente può avvenire solamente in seguito ad una query da parte del client. Per questa ragione sono stati sviluppati degli attacchi in cui si vanno risposte a query che non sono state effettuate.

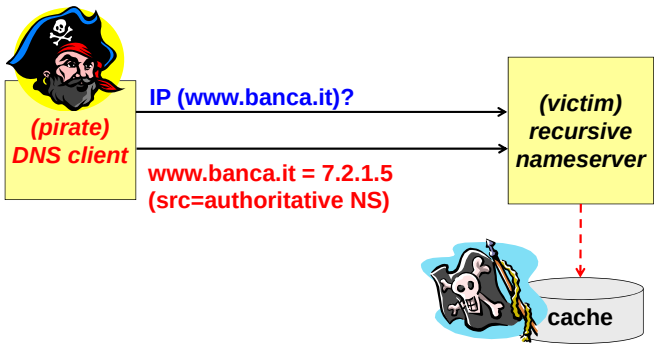


Questi attacchi, che vengono definiti DNS cache poisoning, iniziano con l'attirare la vittima ad effettuare una query sul NS dell'attaccante ("il primo a connettersi su questo server riceverà in regalo uno smartphone"). A questo punto il client andrà a contattare il local NS, chiedendogli di effettuare la traduzione del dominio pirata. Il local NS andrà allora a contattare il NS pirata il quale fornirà la risposta corretta che permetterà all'utente di connettersi al dominio pirata. Tuttavia però nella risposta non sarà solamente presente questa traduzione, ma verranno aggiunte altre coppie nome-indirizzo errate le quali verranno anch'esse aggiunte nella cache.

In questo senso si andranno ad avere risposte a query che l'utente non ha mai effettuato. Non è facile andare ad individuare questo tipo d'attacco, e per tale ragione è stato particolarmente efficace in passato (alcune varianti continuano ancora ad avere successo, proprio perché è difficile andare a capire se la risposta ricevuta è pertinente alla domanda effettuata).

DNS cache poisoning (v2)

In questa seconda variante l'attaccante non aspetta nemmeno che il client effettui la query: è direttamente l'attaccante a fare la domanda e a fornire automaticamente la risposta.



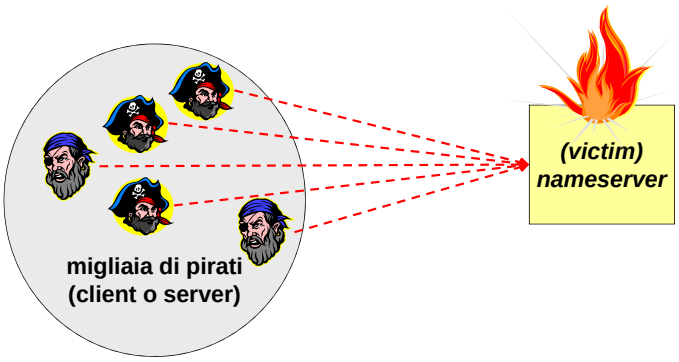
Questa volta quindi il pirata non è più un NS ma è un client; in particolare:

- Esso andrà a fare una domanda al local NS della vittima.
- Il local NS però non è a conoscenza della traduzione, e quindi provvederà a contattare un NS di livello superiore. È a questo punto che l'attaccante invierà la risposta alla domanda precedentemente da lui effettuata, in modo che il local NS consideri questa traduzione corretta. Il trucco sta nel fatto che l'indirizzo di tale risposta non sarà quello dell'attaccante (il local NS non accetterebbe mai una risposta proveniente da un client), ma bensì sarà un indirizzo ottenuto tramite IP spoofing. Più precisamente l'attaccante farà finta di essere l'authorative NS del dominio oggetto della domanda.

La risposta ricevuta verrà inserita nella cache del NS. Questo implica che tutti i client che si rivolgeranno a tale NS riceveranno una risposta sbagliata a seguito di questa traduzione nome-indirizzo errata.

(DNS) flash crowd

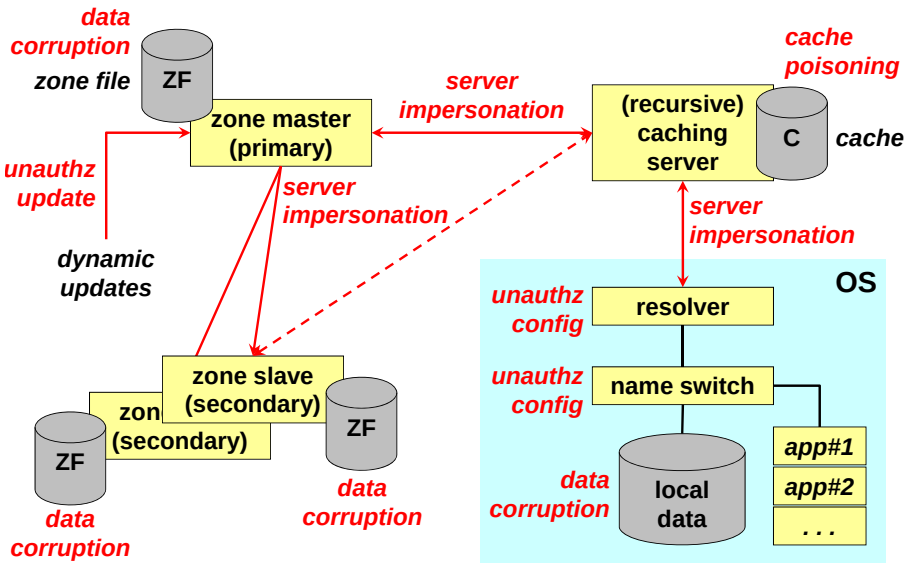
Un attacco di tipo DoS eseguibile su DNS (ma in generale è eseguibile su tanti tipi di servizi) è la cosiddetta flash crowd.



Un attacco di questo genere consiste nel mettersi d'accordo fra tantissimi utenti della rete e fare svolgere a tutti una stessa operazione simultaneamente. Mentre i server applicativi sono abituati ad avere una grossa mole di utenti, i DNS server non sono predisposti a ricevere un numero elevato di domande in un periodo di tempo molto breve.

Questo attacco quindi può andare a rendere indisponibile un servizio non perché non funzionano i server sui quali essi risiedono, ma perché nessun utente riesce a sapere quali siano gli indirizzi di tali server.

Traduzione nomi-indirizzi



Chi chiede di risolvere un nome tipicamente è un'applicazione che risiede su un SO. Normalmente all'interno del SO c'è un piccolo processo chiamato *name switch*, il quale permette di risolvere nomi-indirizzi rivolgendosi o a dati memorizzati in locale (dati presenti all'interno del file hosts) o a dati esterni (nel caso i cui non fosse possibile risolvere il nome usando i dati locali).

Qualora fosse necessario far riferimento a dati esterni questo non avviene direttamente ma tramite un altro processo detto *resolver*, che ha proprio il compito di risolvere la corrispondenza nome-indirizzo. Il resolver viene usato principalmente perché un sistema potrebbe non usare solamente il protocollo DNS; è infatti possibile andare ad usare in reti locali altri sistemi, come ad esempio NIS per i sistemi Linux. Questi sistemi forniscono un servizio di traduzione ma solamente per la rete locale.

Si supponga comunque che il resolver comunichi solamente con il DNS. La prima operazione eseguita è quella di contattare il *local NS*, il quale è un caching server in quanto tiene in memoria tutte le traduzioni precedentemente eseguite in modo da non doverle eseguire di nuovo. Il local NS viene anche definito *recursive* perché una volta che avrà contattato il root NS e questo gli avrà risposto, sarà sempre lui a contattare il NS di primo livello e di livello inferiore fino a quanto non riuscirà a risolvere correttamente il nome.

Il cache server contatta i vari livelli di NS. a ciascun livello è possibile andare a parlare con un *master (primary)* o con uno *slave (secondary)*: ogni dominio di qualunque livello deve avere almeno 2 NS (meglio tre di cui una off-site). Il zone master è il NS autoritativo che contiene lo *zone file*, file che contiene tutti i nomi e gli indirizzi di quella determinata zona. Una copia di questo file viene anche mandata, tramite il trasferimento con canale TCP, a dei server secondari, i cosiddetti *zone slave*. Questo significa che eventuali modifiche ai nomi o agli indirizzi dovranno essere effettuate sullo zone file presente sul primary NS, in quanto gli slave hanno soltanto delle copie. In ogni caso quando si fa una richiesta di traduzione si riceve un elenco di NS a cui è possibile rivolgersi; il caching server inizierà a questo punto a contattare tutti i NS fino a quando non riuscirà ad avere una risposta.

Mentre in passato ogni volta che bisognava aggiungere o cambiare qualcosa nel NS era il sistemista che andava a modificare manualmente lo zone file, in Windows sono state aggiunte le cosiddette *dynamic updates*. Le dynamic updates permettono ai client DNS di registrarsi e aggiornare in modo dinamico le loro associazioni nome-indirizzo presenti su un NS, nel senso che quando un client inizia ad essere usato esso comunica al NS il proprio indirizzo in modo che la sua associazione nome-indirizzo possa essere aggiornata di conseguenza. Questo risulta

soprattutto utile quando un client può muoversi liberamente all'interno di una rete locale, e quindi può modificare il proprio indirizzo.

Questo sistema però sconvolge la struttura del DNS, in quanto non è più vero che le tabelle vengono modificate di rado, ma verranno modificate ogni volta che un client modifichi il proprio indirizzo. Inoltre si avranno anche dei problemi di autorizzazione, in quanto bisogna essere certi che un certo client è effettivamente autorizzato ad andare a cambiare le informazioni sul DNS.

Data la struttura di questo sistema, sono possibili una serie di attacchi:

- Lo user terminal è attaccabile nel senso che se un'attaccante riesce ad avere accesso fisico (o anche solo logico tramite ad esempio una backdoor) potrebbe andare a modificare i dati memorizzati in locale (*data corruption*) oppure andare a modificare i dati di configurazione del name switch o del resolver (*unathorized configuration*). In particolare la modifica della configurazione del resolver è abbastanza semplice da eseguire tramite DHCP, perché tale protocollo non soltanto fornisce ad un client l'indirizzo IP ma fornisce anche il NS da usare per quel collegamento.

- Nel collegamento tra il resolver e il caching server potrebbe esserci uno shadow server (*server impersonation*).
  Sul caching server invece potrebbe essere eseguito un attacco cache poisoning.
  Infine anche i caching server possono essere impersonificati, nel senso che possono ricevere risposte da un uno shadow NS. Uno shadow server in questa posizione ha molta più efficacia rispetto ad averlo nel collegamento locale in quanto provoca una modifica della cache per tutto il dominio locale, mentre nel caso locale la modifica riguarda solamente un singolo nodo.

- Per quanto riguarda lo zone master, se qualcuno riuscisse ad avere accesso fisico (o logico) allo zone file e ad eseguirvi delle modifiche questo causerebbe molti danni, soprattutto perché tali modifiche verrebbero propagate anche agli zone slave. Lo zone file deve quindi essere protetto per integrità, in modo da evitare che qualcuno possa manipolarlo.
  In aggiunta le dynamic updates di Microsoft aggiungono un problema di autenticazione e autorizzazione.

- Gli slave invece hanno il problema dell'impersonation del master, perché se un attaccante si impersonifica come master ed esegue un trasferimento verso gli slave andrebbe a modificare le loro copie dello zone file. Da qui nasce una necessità di avere autenticazione negli zone transfer.
  Infine poiché anche sugli slave è presente una copia locale dello zone file, è possibile che qualcuno riesca ad avervi accesso vada a modificarli a piacimento.

DNSsec: Una necessità

In seguito ai numerosi problemi di sicurezza legati all'uso del DNS, si è cercato negli anni di rendere più sicuro tale protocollo. In particolare si hanno avuto grandi cambianti quando nel febbraio 2008 un ricercatore (Dan Kaminsky) ha scoperto un nuovo attacco che rende il cache poisoning molto più facile da fare, molto più difficile da evitare e addirittura applicabile anche ai record dei NS di primo livello.

Soltanto nel luglio del 2008 escono i primi advisory e le prime patch al problema. A questo punto visto che ormai il problema era diventato noto e che le prime soluzioni iniziavano ad uscire, Kaminsky nell'agosto del 2008 viene autorizzato a rendere pubblico il problema tenendo una conferenza nel Black Hat '08.

In seguito a questo, nel settembre del 2008 gli USA rendono obbligatorio l'uso del DNSsec per il dominio .gov a partire dal gennaio 2009.

DNSsec è un sistema in cui tutti i record che sono inseriti dentro gli zone file sono firmati tramite una firma digitale. La firma digitale fornisce integrità (nessuno può andare a modificare i record a suo piacimento) ma anche autenticazione. Tuttavia questo introduce 2 problemi:

- Quali certificati sono da usare per effettuare queste firme digitali? Quali PKI sono utilizzabili?
- Il soggetto che ha firmato un record è authoritative per un certo dominio?

L'introduzione della firma digitale per ogni record causa una gestione notevolmente più complessa dell'infrastruttura DNS perché si avranno delle firme gerarchiche, ossia quando si riceve una risposta bisognerà controllare tutta una catena di firme, e inoltre occorrerà controllare la validità delle deleghe, nel senso che bisogna essere sicuri che il NS che ha fornito la risposta sia stato effettivamente delegato dal NS di livello superiore. Si ha inoltre il problema delle firme distribuite nel momento in cui si va ad utilizzare un NS master e più slave: da chi dovrà essere firmata la risposta?

Delicata è anche la gestione dei nomi inesistenti, ossia come comportarsi nel caso in cui un si chiede la traduzione di un record inesistente. Quello che viene fatto è andare a firmare anche l'assenza di un record, in modo da evitare possibili attacchi di DoS (un attaccante potrebbe andare a rispondere prima del NS dicendo che quel dominio non è raggiungibile); questo però richiede che i record siano ordinati, e che le risposte generate vengano firmate al volo. (Perché devono essere ordinate? Nel DNS sono presenti i record ordinati www.a.com e www.b.com; un utente richiede la traduzione di www.azzardo.com. Essendo i record ordinati il DNS può dimostrare che non esiste il dominio cercato andando a fornire l'elenco ordinato dei domini esistenti. Per tale ragione le firme devono essere fatte al volo, perché i dati da firmare sono dinamici.

Anche se DNSsec è molto più sicuro rispetto al semplice DNS, il suo uso non è ancora così frequente in quanto presenta alcuni problemi:

- Non si ha nessuna firma delle query DNS, e quindi possono essere generate delle domande false.
- Non esiste nessuna root CA (nel senso che non esiste un'unica root CA, ce ne sono diverse), ma le chiavi dei root NS sono distribuite OOB. Tale elenco di chiavi può quindi essere soggetto ad attacchi.
- Non si ha nessuna sicurezza nel dialogo tra il DNS client e il DNS local server. Si presuppone che questa tratta qui sia stata sicurizzata in qualche altro modo, ad esempio tramite IPsec, TSIG o SiG(O) (quest'ultimi due servono a fare autenticazione e integrità delle domande e delle risposte fra DNS client e DNS local server).
- Necessità di eseguire la crittografia direttamente sui server DNS. Questo però genera un sovraccarico computazionale, ma anche un sovraccarico gestionale (il DNS dovrà diventare un on-line secure crypto host).
- I record avranno una maggiore dimensione.
- Visto l'ancora scarso utilizzo di questa soluzione non si ha ancora avuto una grande sperimentazione, e quindi si ha molta difficoltà a trovare le corrette configurazioni per garantire un certo livello di prestazioni.

5.10.6 Sicurezza del routing

La sicurezza del routing è molto problematica, innanzitutto perché il routing è un'infrastruttura distribuita (i router sono sparsi lungo tutto il territorio), i quali devono essere gestiti remotamente. Purtroppo i protocolli usati per la gestione remota dei router non sono particolarmente sicuri; ad esempio SNMP, usato per il network management, è altamente insicuro, e permette facilmente di prendere possesso e di manipolare dei nodi di rete.

Anche se non fosse possibile attaccare direttamente le apparecchiature, va tenuto conto che i router si scambiano le tabelle di routing. Anche in questo caso si ha un basso livello di sicurezza negli scambi perché tipicamente l'autenticazione è basata sugli indirizzi IP (è sufficiente effettuare un IP spoofing). Esiste la possibilità di proteggere lo scambio di queste tabelle tramite uno

standard che prevede la protezione con keyed-digest, e in questo modo le tabelle scambiate saranno integre ed autentiche. Questa soluzione percò richiede l'uso di una chiave segreta ma condivisa, e quindi si hanno tutti i problemi legati al key-management.

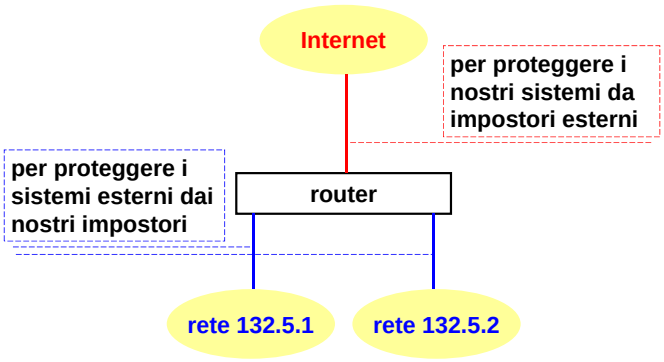
Infine le variazioni di routing possono anche essere fatte sugli end-node tramite l'uso improprio del comando ICMP redirect.

5.10.7 Protezione da IP spoofing

Per molti attacchi visti in precedenza è necessario fare IP spoofing. IP spoofing non può essere eliminato completamente, perché chiunque può mettere un indirizzo falso ad un pacchetto. Si può però cercare di limitare il problema; per fare questo però il gestore di una rete dovrebbe attivare una serie di regole minime per cercare di proteggere gli utenti di quella rete da impostori esterni, ma anche per proteggere il mondo dagli impostori interni alla rete. La prima è una misura di sicurezza pressoché obbligatoria, mentre la seconda è una misura cosiddetta di net-etiquette, ossia di sapersi comportare correttamente in rete nei confronti degli altri nodi.

Queste specifiche di sicurezza sono contenute nell'RFC-2827, che indica come cercare di limitare i DoS basati su IP spoofing, e su altri RFC ausiliari come RFC-3704 e RFC-3013, i quali spiegano come effettuare il filtraggio per reti multihomed e in generale quali sono le misure di sicurezza raccomandate per gli ISP.

Filtri per protezione da IP spoofing



Si immagini di gestire le due reti blu. Queste due reti sono collegate ad un border router, il quale fornisce un collegamento al resto di Internet.

Sul traffico verso internet si vuole cercare di proteggere i sistemi interni alla reti da impostori esterni, ossia si vuole evitare che qualcuno arrivi da internet fornendo come indirizzo uno degli indirizzi interni alle reti blu; non è possibile che un pacchetto proveniente da internet abbia come indirizzo un indirizzo interno ad una delle due reti blu. Questa è il tipo di filtraggio che occorre fare sui pacchetti entranti da Internet.

Per quanto riguarda le misure di net-etiquette sono da implementarsi sulle interfacce interne, perché su tali interfacce sono da accettarsi solamente i pacchetti che hanno come source address un indirizzo contenuto all'interno di una delle due reti blu.

5.10.8 Sicurezza di SNMP

Molti dei problemi legati al routing sono legati all'uso del protocollo SNMP. Esistono diverse versioni di questo protocollo, in particolare la v1 era completamente prima di qualunque livello di sicurezza, la v2 ha una serie di servizi di sicurezza praticamente inutili e la v3 avrebbe dei servizi di sicurezza ma non vengono quasi mai implementati perché sono troppo complessi per le normali apparecchiature di rete.

L'unica cosa che viene normalmente fatta è che i pacchetti vengono autenticati tramite l'inserimento dentro al pacchetto di un segreto condiviso, trasmesso in chiaro; questo segreto prende

il nome di stringa di “community”. A parte questo non esiste nessuna autenticazione del client o dei server e nessuna protezione dei messaggi.

Capitolo 6

Firewall e IDS/IPS

6.1 Firewall

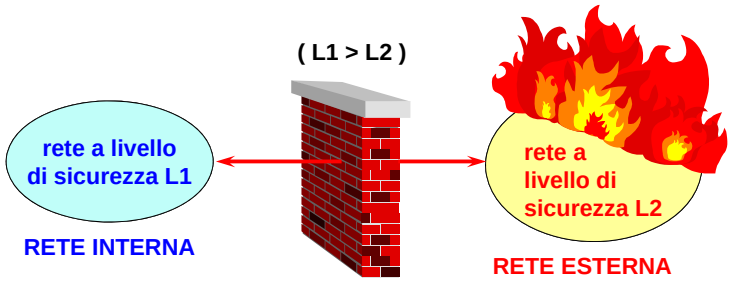


Figura 6.1: Il firewall protegge la rete interna, a un livello di sicurezza L1 maggiore del livello di sicurezza L2 della rete esterna.

firewall un gateway di internetwork che limita il traffico di comunicazione dati da e verso una delle reti connesse (quella che si dice "dentro" il firewall) e così protegge le risorse di sistema di quella rete da minacce dall'altra rete (quella che si dice "fuori" dal firewall)

Il **firewall**, inteso come "muro tagliafuoco" (cioè il muro che limita la propagazione del fuoco da una casa alla casa vicina), è un filtro di rete posto tra due reti che hanno livelli di sicurezza diversi, che serve per garantire la **sicurezza del perimetro**: filtra i pacchetti in transito, lasciando passare solo il traffico "buono", con lo scopo di bloccare la propagazione degli attacchi provenienti dalla rete che si suppone a livello di sicurezza inferiore.

Non necessariamente la rete interna è la rete aziendale e la rete esterna è Internet: il firewall può essere posto anche tra due porzioni di rete entro la stessa azienda, perché gli attaccanti potrebbero essere anche all'interno e vanno protette le aree più sensibili.

I firewall sono efficaci al 100% solo relativamente agli attacchi sui canali che sono bloccati. Per i canali attraverso le porte lasciate aperte occorrono altre difese:

- VPN (sez. 5.8);
- firewall "semantici" (ad es. strong application proxy: sez. 6.3.4);
- IDS (sez. 6.6) e IPS (sez. 6.7);
- sicurezza applicativa (prossimi capitoli).

6.1.1 Modalità

I firewall possono essere configurati in modalità ingress e/o egress:

- **ingress**: controlla le connessioni in entrata, e serve per selezionare quali servizi devono essere resi disponibili all'esterno (ad es. server Web);
- **egress**: controlla le connessioni in uscita, e serve per:
  - impedire ai nodi interni di collegarsi a nodi potenzialmente pericolosi;
  - controllare che l'attività degli impiegati sia in linea con le politiche interne (ad es. filtro Facebook, fuga di segreti aziendali).

La distinzione tra ingress ed egress è:

- facile per i servizi orientati al canale (ad es. TCP): la connessione è stata instaurata da uno dei due party (ad es. three-way handshake);
- difficile per i servizi basati su datagrammi (ad es. UDP, ICMP): non esiste il concetto di connessione.

Ad esempio, l'implementazione del firewall ingress è problematica nel caso del protocollo FTP: nonostante sia l'utente dalla rete interna ad aver aperto la connessione FTP, il canale per il trasferimento di file viene aperto sempre dal server esterno.

6.1.2 Elementi di base

Gli elementi di base che formano un firewall sono:

- **screening router** (choke) (sez. 6.5.1): è un router che, oltre all'instradamento, filtra il traffico a livello IP;
- **bastion host** (sez. 6.2.4): è un sistema particolarmente sicuro e protetto, con funzioni di auditing (ad es. logging);
- **application-level gateway** (proxy) (sez. 6.3.4): è un servizio che svolge il lavoro per conto di una certa applicazione, filtrando il traffico a livello applicazione (ad es. HTTP) e tipicamente effettuando anche un controllo degli accessi;
- **dual-homed gateway** (sez. 6.5.2): è un sistema dotato di due schede di rete, interfacciate su due reti diverse, per svolgere una funzione di ponte tra una rete e l'altra, ma con l'instradamento disabilitato.

6.2 Progettazione di un firewall

I firewall si progettano, non si comprano: un firewall non è un singolo oggetto, ma è composto da vari componenti che si possono comprare. La progettazione ha lo scopo di trovare un compromesso ottimale tra sicurezza e funzionalità, selezionando i giusti componenti e mantenendo i costi al minimo.

Come filtrare bloccando solo il traffico "cattivo" e consentendo solo il traffico "buono"? Il firewall ideale sarebbe quello che taglia completamente il collegamento tra rete interna e rete esterna, ma ciò renderebbe la rete inutile $\Rightarrow$ è necessario trovare un compromesso tra funzionalità e sicurezza: più pacchetti vengono fatti passare, maggiore sarà la probabilità di propagazione degli attacchi.

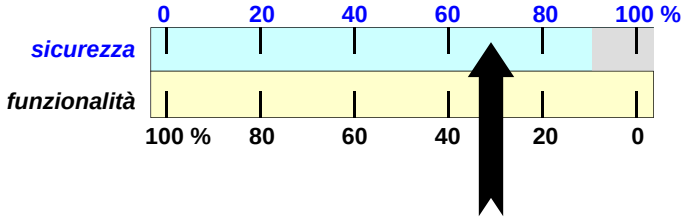


Figura 6.2: L'indice della sicurezza.

6.2.1 I tre comandamenti dei firewall

1. Il firewall deve essere l'unico punto di contatto della rete interna con quella esterna.
   Il perimetro intorno alla rete interna non deve avere delle "brecce" verso il mondo esterno (ad es. un impiegato è connesso allo stesso tempo alla rete aziendale e alla rete mobile con una chiavetta Internet).
2. Solo il traffico "autorizzato" può attraversare il firewall.
   Chi progetta un firewall deve avere una lista dei servizi che devono essere resi disponibili.
3. Il firewall deve essere un sistema altamente sicuro esso stesso.
   Sulla macchina su cui è eseguito il firewall non bisogna installare altro software che potrebbe essere soggetto a bug e vulnerabilità.

6.2.2 Politiche di autorizzazione

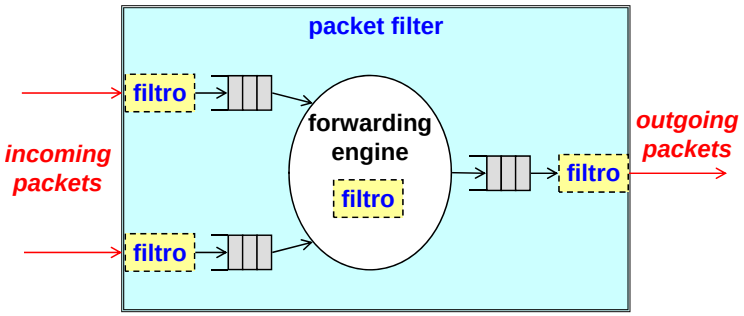
Whitelisting "Tutto ciò che non è espressamente permesso, è vietato"
Una whitelist elenca i servizi autorizzati, e tutti i servizi non inclusi nella whitelist sono bloccati:

- maggior sicurezza: un tipo di attacco imprevisto non avrà successo;
- più difficile da gestire: bisogna capire le reali esigenze degli utenti per evitare di bloccare i servizi essenziali per l'azienda.

Blacklisting "Tutto ciò che non è espressamente vietato, è permesso"
Una blacklist elenca i servizi non autorizzati, e tutti i servizi non inclusi nella blacklist sono autorizzati:

- minor sicurezza: è difficile prevedere tutti gli attacchi possibili;
- più facile da gestire: gli utenti sono più liberi.

6.2.3 Punti di filtraggio



Un forwarding engine elabora i pacchetti accodati in ingresso e li inoltra nelle code in uscita. La scelta della posizione di filtraggio influenza le prestazioni del firewall:

- pacchetti in ingresso: solitamente è la posizione migliore:
  - + filtraggio immediato: il forwarding engine deve elaborare meno dati in ingresso;
  - − più code in ingresso: servono più processi di filtraggio in parallelo;
- forwarding engine: non è così male:
  - + più informazioni: va bene se il filtro è basato più su regole di instradamento che su indirizzi sorgente e di destinazione;
- pacchetti in uscita: generalmente è la posizione peggiore:
  - − risorse computazionali: tutti i pacchetti devono essere elaborati dal forwarding engine.

6.2.4 Bastion host

bastion host un computer fortemente protetto che sta in una rete protetta da un firewall (o è parte di un firewall) ed è l'unico host (o uno di solo alcuni) nella rete che può essere acceduto direttamente dalle reti dall'altra parte del firewall

Il bastion host deve essere un sistema particolarmente sicuro e protetto:

- macchina dedicata: il bastion host non deve essere usato dagli utenti o per installare altro software, ma deve essere adibito esclusivamente alla funzione di firewall;
- software:
  - configurazione minima: è meglio lasciare solo i servizi essenziali a garantire il corretto funzionamento del firewall, senza eseguire processi inutili che potrebbero avere dei bug sfruttabili per compiere attacchi;
  - keep it simple (KISS): il numero di bug cresce esponenzialmente con il numero di righe di codice ⇒ è meglio suddividere in tanti componenti più piccoli, più facili da progettare, implementare e monitorare;
  - solo componenti certificati: software sconosciuto potrebbe contenere malware;
- auditing: deve salvare il log di tutte le attività, meglio se su un server remoto sicuro all'interno della rete e destinato solamente a questo scopo;
- no source routing: la sorgente non deve influenzare i percorsi dei pacchetti, altrimenti potrebbero aggirare il blocco;
- no IP forwarding: solo i pacchetti correttamente controllati e filtrati devono passare da un'interfaccia all'altra;
- trappole per gli intrusi: per esempio la funzione `ls` è modificata in modo da inviare un allarme all'amministratore di sistema.

6.2.5 DMZ

zona buffer un segmento di Internetwork neutrale usato per connettere altri segmenti che operano ciascuno sotto una diversa politica di sicurezza

In una rete aziendale ci potrebbero essere dei server che offrono dei servizi pubblici agli utenti della rete esterna (ad es. server che ospita il sito Web dell'azienda). Se questi server pubblici venissero messi dentro la rete interna, un attaccante dalla rete esterna potrebbe sfruttare una vulnerabilità sul server (ad es. bug nei moduli PHP della pagina Web) per guadagnare l'accesso diretto alla rete interna.

[1] Questa immagine è tratta da Wikimedia Commons (DMZ network diagram 1.svg), è stata realizzata dall'utente Sangre viento, da Jason Funk e dall'utente Pbroks13 e si trova nel dominio pubblico.

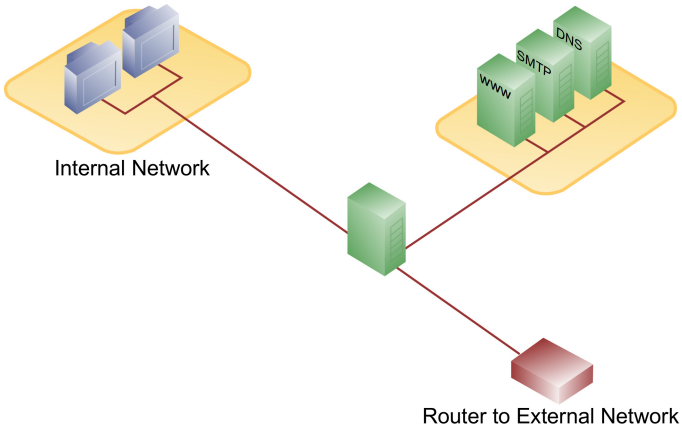


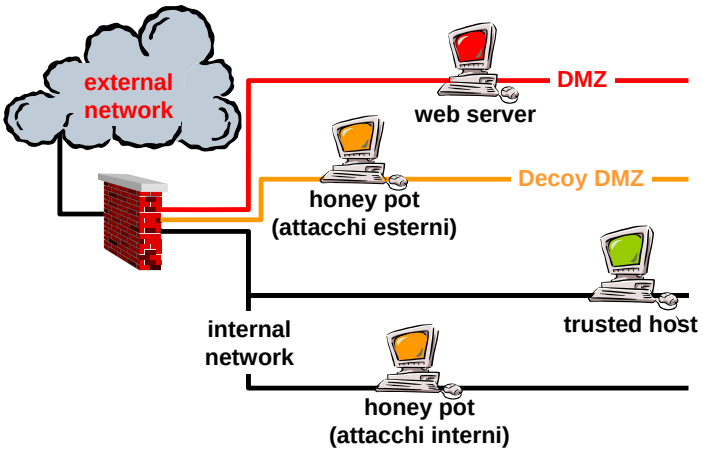
Figura 6.3: Firewall a tre gambe.[1]

La soluzione è mettere tutti i server che devono essere raggiungibili dall'esterno non nella rete interna, ma in un segmento di rete isolato sia dalla rete interna sia da quella esterna, chiamato **De-Militared Zone** (DMZ):

- un eventuale attacco non si propaga alla rete interna dove ci sono gli host;
- i server pubblici sono comunque protetti dal firewall;
- si può configurare l'instradamento in modo che la rete interna sia completamente sconosciuta alla rete esterna.

In commercio sono disponibili firewall dotati di anche più di tre gambe: le gambe ulteriori sono usate per avere più DMZ, in modo da ulteriormente segmentare gli oggetti e impedire a server che svolgono funzioni diverse di comunicare direttamente (ad es. una DMZ per il server della contabilità, un'altra per il server della progettazione).

Honey pot



honey pot un sistema (ad es. un server Web) o una risorsa di sistema (ad es. un file su un server) che è progettato per attirare potenziali cracker e intrusi, come il miele è appetitoso per gli orsi

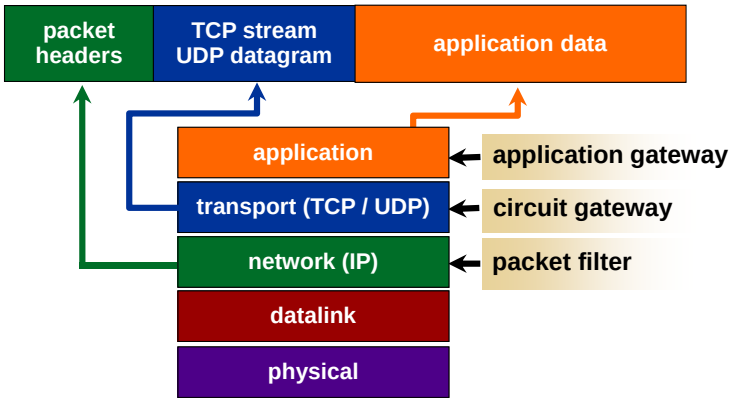
Molte aziende, oltre ad avere una DMZ normale, hanno una seconda DMZ, spesso chiamata **decoy DMZ**, sulla quale sono posti dei server fasulli che sono apposta degli obiettivi facili di

attacco (ad es. nome utente = “root”, password = “root”). Lo scopo è attirare gli attaccanti in modo da analizzare i loro comportamenti e riuscire anche a scoprire la loro identità.

Il firewall può avere dei buchi verso il “miele”. Si può anche mettere un honey pot nella rete interna contro gli attacchi provenienti dall’interno.

Honey net La honey net è un concetto ancora più ampio: viene creata un’intera rete aziendale finta al solo scopo di raccogliere informazioni. Molto spesso queste reti sono create dagli sviluppatori di antivirus: gli attaccanti, infettando queste reti controllate, forniscono informazioni sulle nuove modalità di attacco e i nuovi malware.

6.3 Classificazione dei firewall



A quale livello si fanno i controlli?

- più si scende di livello, maggiore è la velocità del filtraggio: l’elaborazione è più semplice (a livello 3 viene elaborato un pacchetto per volta);
- più si sale di livello, maggiore è la sicurezza: sono disponibili più informazioni su cui prendere le decisioni di filtraggio.

Dal punto di vista commerciale esistono vari flavor classificabili in tre tipi di firewall:

- **packet filter** (sez. 6.3.1): il filtraggio si può basare solo sull’intestazione di ogni pacchetto (livello rete);
- **circuit-level gateway** (sez. 6.3.3): il filtraggio si basa sul flusso TCP/UDP (livello trasporto);
- **application-level gateway** (sez. 6.3.4): il filtraggio è basato sul payload (livello applicazione).

Dal punto di vista tecnico, le differenze principali tra questi sistemi sono in termini di:

- prestazioni;
- protezione del sistema operativo del firewall stesso;
- fedeltà al modello client-server.

Quando due apparati di sicurezza (ad es. un packet filter e un gateway) sono posti in serie, è bene che siano di produttori diversi: se il software del primo apparato avesse un baco, molto probabilmente lo avrebbe anche il secondo se entrambi si basano su librerie comuni $\Rightarrow$ a un attaccante basta sfruttare questo baco per aggirare entrambi gli apparati.

6.3.1 Packet filter

Il **packet filter**, storicamente disponibile sui router, effettua controlli sui singoli pacchetti IP in base a:

- intestazione IP: indirizzi;
- intestazione TCP/UDP: porte.

Vantaggi

- ottima scalabilità: un packet filter si limita ad analizzare le intestazioni indipendentemente dai protocolli applicativi;
- ottime prestazioni: è implementabile in hardware;
- basso costo: è una funzionalità disponibile su tutti i router e su molti sistemi operativi.

Svantaggi

- stateless: ogni pacchetto è analizzato indipendentemente dagli altri;
- frammentazione IP: nel pacchetto potrebbe non esserci l'intestazione TCP/UDP;
- sicurezza: i controlli sono poco precisi e quindi più facili da ingannare (ad es. IP spoofing);
- configurazione complessa: è difficile avere a che fare con dei numeri invece che con dei nomi;
- si hanno dei problemi a supportare dei servizi basati sull'allocazione dinamica delle porte (ad es. FTP).

6.3.2 Packet filter stateful (dinamico)

Il **packet filter stateful** (dinamico) è 'state-aware':

- memorizza delle informazioni di stato dal livello trasporto e/o dal livello applicativo;
- distingue le nuove connessioni da quelle già aperte, grazie a una tabella di stato per le connessioni aperte ⇒ i pacchetti che corrispondono ad una riga della tabella sono accettati senza ulteriori controlli.

Esempio Il packet filter apre temporaneamente la porta FTP solo quando inizia un trasferimento file con il comando `PORT`.

Vantaggi

- stateful: le decisioni sono migliori perché basate su informazioni di stato;
- parallelizzazione: più core della CPU lavorano in parallelo (Symmetrical Multi-Processing [SMP]).

Svantaggio Rimangono molte limitazioni proprie del packet filter.

6.3.3 Circuit-level gateway

Il **circuit-level gateway** è un proxy non "application-aware":

- crea un circuito di livello trasporto tra il client e il server;
- non ha alcuna comprensione della sintassi dei dati in transito.

Vantaggio I server sono isolati da tutti gli attacchi che riguardano:

- il three-way handshake TCP: la protezione si attiva all'inizio della sessione e dura per tutta la sessione;
- la frammentazione dei pacchetti IP: il proxy riassembla il pacchetto per comprendere completamente il suo contenuto.

Svantaggi

- rottura del modello client-server (per la durata della sessione): può richiedere modifiche alle applicazioni (ad es. autenticazione del client: avviene normalmente a livello applicativo, non a livello trasporto);
- rimangono molte limitazioni proprie del packet filter.

6.3.4 Application-level gateway

L'**application-level gateway**:

- ispeziona i pacchetti a livello applicativo (payload): è composto al suo interno da una serie di proxy, uno per ogni protocollo applicativo;
- può svolgere il ruolo di terminatore: interagisce direttamente con il client come se fosse il server, e con il server come se fosse il client;
- può effettuare il mascheramento o la rinumerazione degli indirizzi IP: può essere necessario se sta agendo da terminatore e uno dei due party chiede l'autenticazione dell'altro;
- può anche avere funzioni di autenticazione, soprattutto in egress: il gateway chiede al client nella rete esterna di autenticarsi in modo da applicare a esso le politiche appropriate.

Vantaggi

- massima sicurezza: le regole sono più granulari e più semplici rispetto al packet filter, in quanto i controlli si basano sul livello applicativo;
- parallelizzazione: più core della CPU lavorano in parallelo (SMP).

Svantaggi

- prestazioni: controlli più approfonditi richiedono più tempo;
- dipendente dalle applicazioni: ogni protocollo applicativo richiede uno specifico proxy:
  - ritardo nel supporto di nuove applicazioni;
  - consumo di risorse: molti proxy significano molti processi;
  - basse prestazioni: i processi lavorano in user mode;
- rottura del modello client-server: può non essere del tutto trasparente ai client, e richiede spesso una modifica dell'applicativo client;
- attacchi: poiché il client interagisce direttamente con il gateway, il sistema operativo del firewall è esposto ad attacchi;
- protocolli di sicurezza di livello applicativo: il firewall potrebbe non essere in grado di interpretare correttamente il contenuto dei pacchetti (ad es. SSL: il contenuto del pacchetto è criptato).

Varianti

- **transparent gateway**:
  - \+ trasparente: è meno intrusivo perché non richiede configurazione sul client;
  - − complessità di implementazione: l'intelligenza del sistema viene spostata dal client al router, che dovrà fare il reinstradamento dei pacchetti verso i proxy;
- **strong application proxy**: i controlli sono basati sulla semantica e sulla politica, non solo sulla sintassi del protocollo applicativo:
  - sintassi: il comando `GET` è nel formato di protocollo corretto?
  - semantica: il comando `GET` chiede una risorsa effettivamente esistente?
  - politica: il comando `GET` chiede una risorsa autorizzata dalla politica?

6.3.5 Reverse proxy

Il **reverse proxy** è un filtro ingress posto subito davanti a uno o più server HTTP: fa solo da front-end verso il client, fingendosi il server, e poi passa le richieste al vero server.

Funzionalità

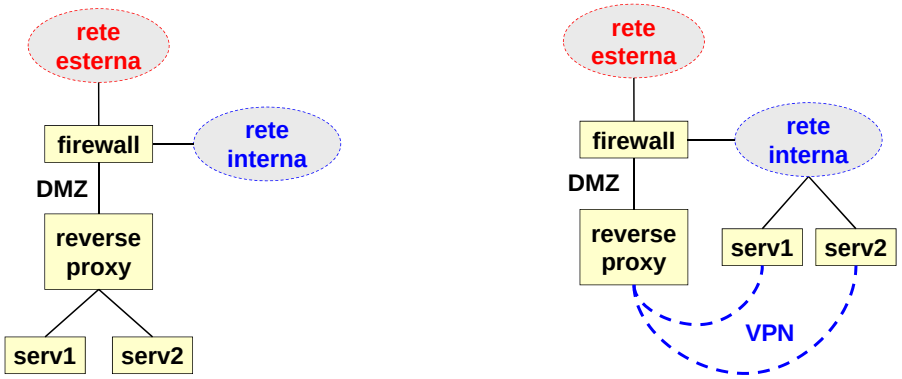
- ispezione dei contenuti: controlla ogni richiesta prima di passarla al server;
- Access Control List (ACL): decide quali indirizzi possono collegarsi e quali protocolli possono essere utilizzati;
- obfuscation: il proxy non dichiara il vero tipo di server utilizzato, proteggendo da alcuni tipi di attacco;
- acceleratore SSL: diminuisce il numero di operazioni crittografiche che il server deve effettuare migliorando le prestazioni, ma le comunicazioni tra il reverse proxy e il server saranno non protette (almeno a livello di protocollo);
- load balancer: seleziona il server meno carico, garantendo una migliore resistenza agli attacchi DoS;
- web accelerator: memorizza localmente in cache i contenuti statici richiesti più spesso, così da diminuire il carico sul server;
- compressione;
- spoon feeding: copia in locale un'intera pagina creata dinamicamente, e la invia poco per volta al client ⇒ è utile quando l'utente è connesso a una rete molto lenta (ad es. rete mobile) per scaricare il server applicativo.

Architettura

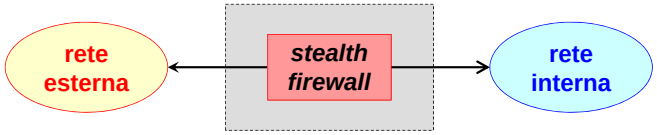
Dal punto di vista architetturale, il reverse proxy è posto obbligatoriamente sul canale DMZ perché sarà raggiungibile dall'esterno.

E se i server hanno bisogno di accedere a un database nella rete interna? Sono possibili due configurazioni:

- i server stanno nella DMZ, fisicamente attaccati al proxy, e hanno dei canali di ritorno verso il database nella rete interna ⇒ gli accessi al database sono molto lenti;
- i server stanno nella rete interna, quindi più vicini al database, e il proxy comunica con i server attraverso una VPN (ad es. IPsec end-to-end).



6.3.6 Stealth firewall



Lo **stealth firewall** è un firewall privo di un indirizzo di rete: intercetta fisicamente i pacchetti in transito, mettendo le proprie interfacce di rete in modalità promiscua, e se soddisfano la politica di sicurezza li inoltra, assolutamente inalterati, all'altra interfaccia di rete.

La sua presenza è invisibile al client: non può essere destinatario di alcun pacchetto, quindi non è attaccabile direttamente. Al massimo si può notare un aumento del ritardo di trasmissione, a seconda del tempo richiesto dai controlli.

Non tutti i firewall possono essere utilizzati in modalità stealth:

− application-level gateway: deve avere un indirizzo essendo basato su una comunicazione client-server;

− circuit-level gateway: deve avere un indirizzo dovendo svolgere la funzione di terminatore;

\+ packet filter: deve solo guardare dentro i pacchetti.

6.3.7 Local firewall e personal firewall

Il firewall può essere installato direttamente sul nodo da difendere:

- **local firewall**: è un firewall, soprattutto ingress, installato sul server;
- **personal firewall**: è un firewall, soprattutto egress, installato sul client.

Essendo installato direttamente sul nodo da proteggere, gira all'interno del sistema operativo e quindi può controllare non solo i protocolli e le porte, ma anche a quali processi è permesso:

- agire da client (egress): aprire collegamenti in rete verso altri nodi (ad es. un malware può inviare dei dati a nodi remoti e propagarsi ad essi);
- agire da server (ingress): ricevere richieste di collegamento o di servizio.

6.4 SOCKS

SOCKS, anche noto come "Authenticated Firewall Traversal" (AFT), è il circuit-level gateway più diffuso al mondo. Il grosso successo di SOCKS deriva dal fatto che è supportato commercialmente sia dai browser (ad es. Firefox, Internet Explorer) sia da diversi firewall (ad es. IBM).

Per poter autenticare i client a livello trasporto, i client devono essere modificati con la libreria open source di SOCKS:

- contiene i client standard (ad es. Telnet, FTP, Finger, Whois);
- fornisce una libreria per sviluppare i propri client.

Lato client La libreria rimpiazza le funzioni standard che manipolano i socket (`connect()`, `bind()`, `accept()`, ...) con funzioni che hanno lo scopo di:

- aprire un canale con il server SOCKS;
- inviare:
  - il numero di versione del protocollo;
  - l'indirizzo IP e la porta a cui il client intende collegarsi;
  - il nome utente dell'utente che sta richiedendo l'operazione.

Lato server Il server SOCKS:

- controlla la sua ACL per sapere se l'utente ha i diritti necessari ad aprire un collegamento verso l'indirizzo e la porta specificati;
- in caso affermativo, apre il canale richiesto, ma con il proprio indirizzo IP $\Rightarrow$ il server SOCKS riceve i dati dal client, li riassembla e li inoltra verso il server di destinazione (viceversa per le risposte).

6.4.1 SOCKS 5

SOCKS 4 presentava i seguenti problemi:

- non riusciva a distinguere la rete interna da quella esterna;
- l'autenticazione degli utenti era molto debole (basata su un demone chiamato `identd` che accettava la dichiarazione del client);
- supportava solo il protocollo TCP.

La versione 5, la prima sviluppata all'interno dell'IETF, ha apportato diversi miglioramenti:

- supporto UDP;
- distinzione tra rete interna e rete esterna;
- migliore autenticazione (nome utente e password, o sistema simile a Kerberos);
- canale criptato tra client e server SOCKS.

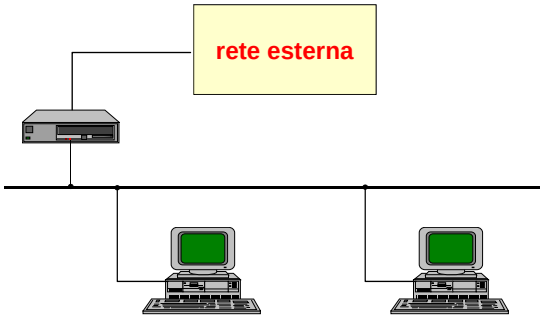
6.5 Architetture di firewall

6.5.1 Screening router (choke)

> **screening router** un router Internetwork che impedisce selettivamente il passaggio di pacchetti di dati secondo una politica di sicurezza

Un singolo apparato, chiamato **screening router**, viene posto tra la rete esterna e la rete interna, svolgendo le funzioni di:

- router: instradamento;
- packet filter: filtraggio del traffico a livello di singoli pacchetti (indirizzi, porte, protocolli).



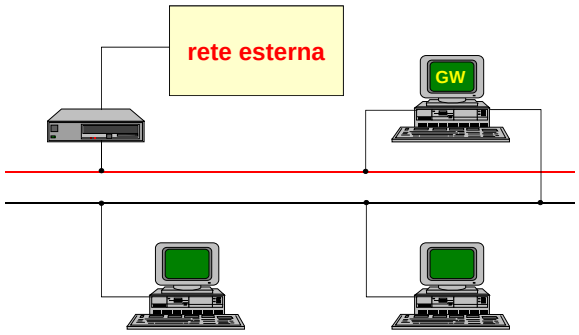
Vantaggi

- no proxy: non richiede modifiche degli applicativi;
- semplicità: è l'architettura più semplice;
- costo: non richiede hardware dedicato.

Svantaggi

- sicurezza: controlla solo a livello rete;
- singolo punto di guasto: il firmware del router potrebbe avere dei bug;
- no DMZ: i server pubblici potrebbero essere fonte di attacco nella rete interna.

6.5.2 Dual-homed gateway



Allo screening router è messo in serie un **dual-homed gateway**, una macchina dotata di due schede di rete, una interfacciata sulla rete esterna e l'altra interfacciata su quella interna, entrambe con instradamento disabilitato, e un processo in esecuzione che ha il compito di decidere quale traffico è autorizzato a passare da un'interfaccia di rete all'altra.

Questo sistema ha un doppio punto di controllo:

- il router svolge la funzione di packet filter;
- il dual-homed gateway ospita un circuit-level gateway o un application-level gateway a seconda delle necessità.

Siccome il gateway si interfaccia su due reti diverse, la rete intermedia è isolata dalla rete interna e può essere usata come DMZ per i server pubblici.

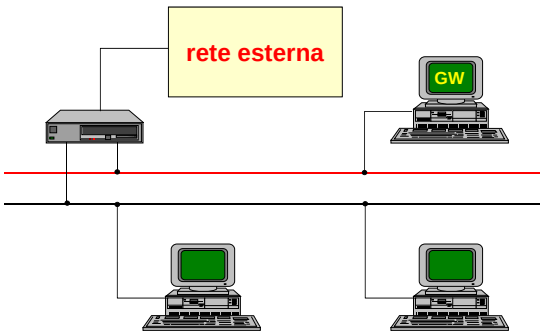
Vantaggi

- mascheramento: il gateway può mascherare la rete interna;
- semplicità: l'implementazione è semplice;
- costo: solo piccoli requisiti hardware aggiuntivi sono richiesti (ad es. scheda di rete veloce);
- doppia linea di difesa: non basta sfruttare il bug di uno dei due dispositivi per entrare nella rete interna.

Svantaggi

- gestione: sono necessari due sistemi;
- collo di bottiglia: tutto il traffico deve passare dal dual-homed gateway;
- flessibilità: alcuni tipi di traffico richiedono il controllo da parte di server nella rete interna (ad es. messaggi di posta elettronica di spam).

6.5.3 Screened host gateway



Il ponte tra la rete interna e la rete esterna è spostato sullo screening router, a cui è stata aggiunta una scheda di rete: una volta che un pacchetto proveniente dall'esterno è autorizzato dal packet filter, esso può prendere due strade:

- può essere inviato ancora al gateway (ora diventato un bastion host), se ha bisogno di ulteriore indagine;
- può entrare direttamente nella rete interna, se ha bisogno di essere controllato da un server interno (ad es. server di posta).

Vantaggio flessibilità: è possibile alleggerire i controlli relativi ad alcuni servizi o host evitando di passare dal gateway.

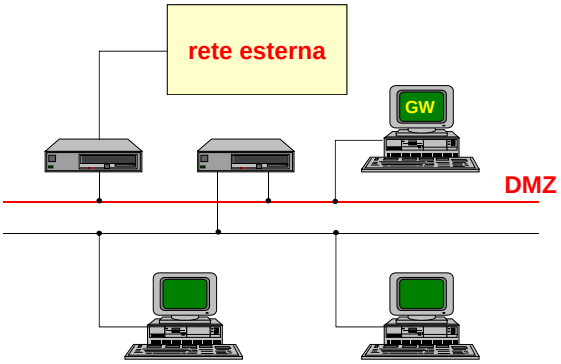
Svantaggi

- mascheramento: il mascheramento è possibile solo per i pacchetti che passano dal bastion host;
- singolo punto di guasto: il firmware del router potrebbe avere dei bug.

6.5.4 Screened subnet

Lo screening router è stato diviso in due:

- la parte di packet filter filtra il traffico in ingresso e in uscita;
- la parte di router svolge la funzione di ponte tra le due reti.



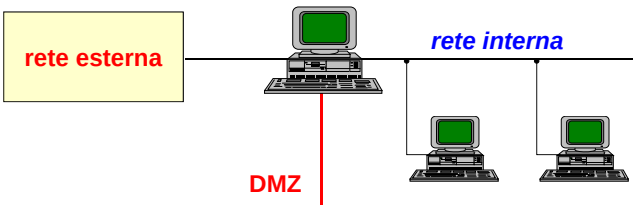
Vantaggi

- flessibilità: è possibile alleggerire i controlli relativi ad alcuni servizi o host evitando di passare dal gateway;
- doppia linea di difesa: non basta sfruttare il bug di uno dei due dispositivi per entrare nella rete interna.

Svantaggi

- costo: sono necessari tre apparati;
- multi-vendor: gli apparati devono essere di tre produttori diversi, altrimenti potrebbero soffrire di bug comuni.

6.5.5 Screened subnet con firewall a tre gambe



Per semplificare l'implementazione dell'architettura screened subnet pur mantenendone la funzionalità, molte aziende hanno proposto di incorporare le funzioni dei due router/packet filter all'interno del gateway. Il gateway ha tre schede di rete, interfacciate ciascuna su tre reti diverse (**firewall a tre gambe**): la rete esterna, la rete interna e la DMZ.

Vantaggi costo minore: su un singolo apparato fisico sono concentrate tre funzioni.

Svantaggio singolo punto di guasto: tre processi sono in esecuzione nello stesso sistema operativo.

6.6 IDS

rilevamento delle intrusioni sentire e analizzare eventi di sistema allo scopo di notare (cioè diventare consapevoli di) tentativi di accedere alle risorse del sistema in una maniera non autorizzata

Intrusion Detection System (IDS) sistema per identificare individui che usano un

computer o una rete senza autorizzazione

Un **Intrusion Detection System** (IDS) monitora la rete per:

- identificare utenti non autorizzati;
- identificare utenti autorizzati, ma che violano i loro privilegi.

Come distinguere i pacchetti di attacco dai pacchetti normali? Il rilevamento si basa sul "pattern" di comportamento: si assume che un attaccante si comporti in modo diverso da un utente normale.

6.6.1 Classificazione

Caratteristiche funzionali

Gli IDS si possono distinguere in base alle caratteristiche funzionali:

- **passivi**: hanno un approccio di tipo reattivo: rilevano un'intrusione dopo che questa è avvenuta:
  - checksum crittografiche: il file è stato modificato?
  - pattern ("attack signature"): il file contiene la firma del virus?
- **attivi**: hanno un approccio di tipo proattivo: rilevano un'intrusione mentre è in corso:
  1. apprendimento: analisi statistica del funzionamento del sistema per apprendere il comportamento normale del sistema (ad es. distribuzione del traffico nei vari giorni della settimana, nelle varie ore del giorno, ecc.);
  2. monitoraggio: analisi attiva del funzionamento del sistema, e confronto con il comportamento previsto dalla statistica alla ricerca di anomalie (ad es. aumento della percentuale di pacchetti ICMP);
  3. reazione: scatta una reazione al superamento di una certa soglia (ad es. eccetto di pacchetti ICMP): potrebbe trattarsi di un attacco oppure no, a seconda anche di dove è stata posizionata la soglia.

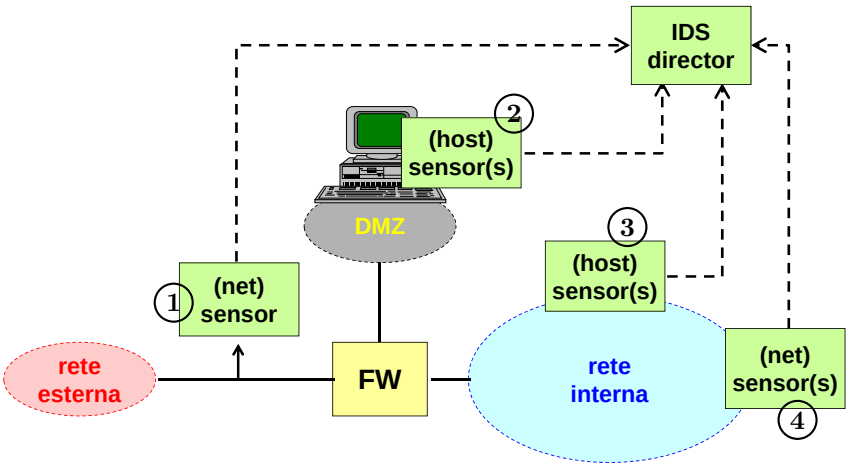
Caratteristiche topologiche

Gli IDS si possono distinguere in base alle caratteristiche topologiche:

- **host-based** (HIDS): si basano sull'attivazione di strumenti di monitoraggio interni al sistema operativo:
  - System Integrity Verifier (SIV) (ad es. Tripwire): controlla il file system per rilevare modifiche ai file (ad es. registri di Windows, configurazione di cron [schedulatore di Linux], cambio privilegi di un utente);
  - Log File Monitor (LFM) (ad es. Swatch): analisi dei log del sistema operativo o anche delle applicazioni, per rilevare pattern conosciuti derivanti da attacchi o tentativi di attacco (ad es. l'utente ha fallito n volte la password);
- **network-based** (NIDS): si basano sull'attivazione di strumenti di monitoraggio del traffico di rete:
  - sensori: sono distribuiti in giro per la rete da monitorare:
    - controllano il traffico e fanno il log individuando i pattern sospetti;
    - attivano i security event in caso di anomalie;
    - possono interagire con il sistema (ad es. mandando dei TCP reset per chiudere i canali, cambiando le ACL);

- director: è il nodo centrale che aggrega le informazioni provenienti dai sensori:
  - coordina l'attività dei singoli sensori;
  - decide se l'anomalia è un attacco in base ai dati nel security database;
- IDS message system: consente una comunicazione sicura ed affidabile tra i componenti dell'IDS.

6.6.2 Architettura di un NIDS



Dove mettere i sensori?

1. sensore di rete prima del firewall: fornisce informazioni sugli attacchi in generale, anche quelli che saranno bloccati dal firewall ⇒ è utile per sapere informazioni statistiche sugli attacchi che il firewall è in grado di bloccare;
2. sensore/i di host nella DMZ: servono per monitorare i server pubblici ospitati sulla DMZ;
3. sensore/i di host nella rete interna: servono per monitorare i server critici (ad es. contabilità, progettazione) nella rete interna;
4. sensore/i di rete nella rete interna: servono per:
   - rilevare se qualcuno dall'esterno è riuscito ad aggirare il firewall;
   - controllare i comportamenti anomali degli impiegati nella rete interna.

6.6.3 Interoperabilità di IDS/NIDS

Non necessariamente tutti i componenti di un IDS arrivano da uno stesso fornitore ⇒ sono necessarie delle soluzioni comuni:

- attack signature: attualmente non esiste alcuno standard per il formato di signature, ma è molto diffuso il formato di Snort;
- allarmi: sono in competizione due soluzioni per il formato e il protocollo di trasmissione degli allarmi:
  - IDMEF + IDXP + IODEF: tre protocolli standard proposti da IETF;
  - SDEE: protocollo spinto da alcune aziende (ad es. Cisco).

6.7 IPS

Intrusion Prevention System (IPS) sistema che può rilevare un'attività intrusiva e può anche provare a fermare l'attività, idealmente prima che essa raggiunga i suoi obiettivi

Un **Intrusion Prevention System** (IPS) è un sistema automatico che serve per velocizzare ed automatizzare la risposta alle intrusioni:

- IDS: si limita ad avvisare che è in corso un attacco;
- IPS: tenta di prevenire gli attacchi.

Un IPS è costituito da:

- IDS: rileva un attacco in corso (ad es. eccesso di pacchetti ICMP);
- firewall dinamico distribuito: reagisce molto velocemente (ad es. blocca tutto il traffico ICMP).

Svantaggi

- siccome le decisioni sono automatiche, un IPS potrebbe prendere decisioni sbagliate bloccando del traffico innocuo;
- non è un prodotto ma una tecnologia, con un grosso impatto su tanti elementi del sistema di protezione.

Capitolo 7

Sicurezza delle applicazioni di rete

Tipicamente gli sviluppatori di applicazioni di rete non considerano alcuni aspetti di sicurezza:

- problemi di autenticazione debole:
  - password snooping: quando l'autenticazione è basata solo su nome utente e password senza l'utilizzo di canali protetti;
  - IP spoofing: quando l'autenticazione è basata sugli indirizzi IP;
- altri problemi frequenti:
  - data spoofing: i dati possono essere letti;
  - data forging: i dati possono essere manipolati;
  - attacchi shadow server e man-in-the-middle;
  - attacchi replay.

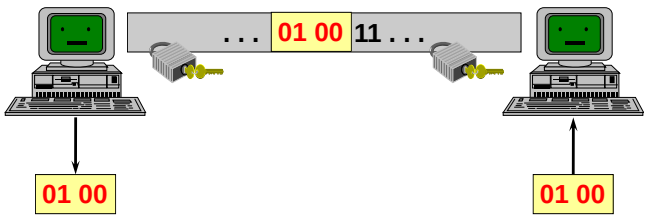
7.1 Approcci di sicurezza

Per cercare di risolvere questi problemi di sicurezza si possono adottare due approcci, teoricamente opposti ma in realtà complementari:

- sicurezza di canale (sez. 7.1.1): tutti i dati sono protetti solo durante il transito nel canale;
- sicurezza di messaggio (sez. 7.1.2): solo i dati imbustati sono sempre protetti.

I due approcci possono essere combinati trasmettendo dati sensibili protetti su canali sicuri.

7.1.1 Sicurezza di canale



La sicurezza è realizzata a livello di canale di rete: prima della trasmissione i due peer instaurano un canale sicuro (ad es. IPsec, TLS), e tutti i dati che passeranno per quel canale saranno protetti, anche se non sono stati incapsulati in buste sicure.

Proprietà di sicurezza

+ integrità: se i dati sono stati alterati, il ricevitore non riuscirà a capirli nella decriptazione;
+ autenticazione (singola o mutua): il canale viene negoziato tra i due peer;
+ riservatezza: il canale è criptato;
− non ripudio: i dati non possono essere firmati digitalmente dal mittente.

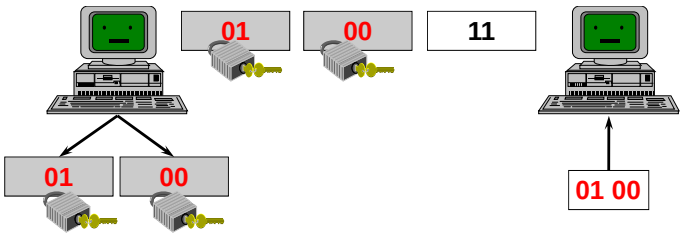
Vantaggi Richiede al più piccole modifiche alle applicazioni:

- IPsec: non richiede alcuna modifica in quanto è installato a livello IP;
- TLS: le chiamate ai socket devono essere cambiate in chiamate a TLS.

Svantaggi

- protezione forzata: tutti i dati in transito sono protetti, anche quelli che non hanno bisogno di protezione;
- protezione temporanea: i dati sono protetti solamente durante il transito nel canale, e ritornano a essere insicuri quando escono dal canale;
- non ripudio: i due peer devono fidarsi l'uno dell'altro.

7.1.2 Sicurezza di messaggio



La sicurezza è realizzata a livello di dati: i dati sensibili sono incapsulati in buste sicure (ad es. PKCS #7[1]), e possono essere inviati anche su canali non sicuri.

+ integrità: il ricevente potrà rilevare un'alterazione per mezzo di vari meccanismi (ad es. firma digitale non valida, keyed-digest non corrispondente);
- autenticazione:
  + singola: il destinatario riceve dal mittente delle buste su cui possono essere stati applicati vari meccanismi (ad es. firma digitale, keyed-digest) che forniscono l'autenticazione del mittente;
  − mutua: il destinatario non invia al mittente alcun dato su cui applicare dei meccanismi che forniscano l'autenticazione;
+ riservatezza: le buste sono criptate;
+ non ripudio: le buste sono firmate digitalmente dal mittente.

[1] Si veda la sezione 3.10.

Vantaggi

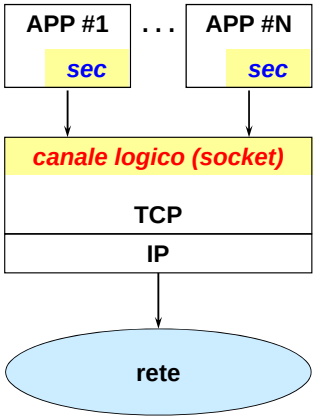
- protezione non forzata: possono essere protetti solo i dati effettivamente sensibili, mentre i dati non sensibili verranno inseriti in normali pacchetti;
- protezione permanente: i dati rimangono protetti anche fuori dal canale (ad es. memorizzati sul disco);
- non ripudio: il mittente non può negare di aver digitalmente firmato i dati.

Svantaggi Richiede modifiche alle applicazioni:

- lato mittente: l'applicazione che genera/invia i dati deve incapsularli in una busta;
- lato ricevitore: l'applicazione che riceve/elabora i dati deve estrarli dalla busta.

7.2 Implementazione dei sistemi di sicurezza

7.2.1 Sicurezza interna alle applicazioni



Ogni applicazione ha la sua parte di sicurezza direttamente implementata al proprio interno dallo sviluppatore. La parte in comune tra le varie applicazioni si limita ai canali di comunicazione (socket).

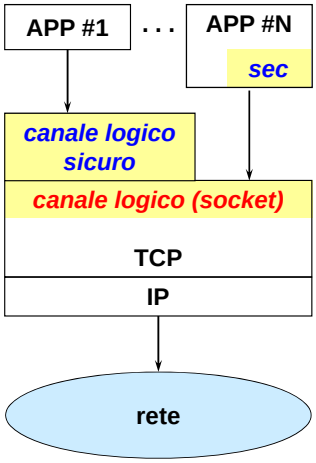
Vantaggio sicurezza di messaggio: solo l'applicazione conosce il formato dei dati trasmessi e ricevuti

Svantaggi

- errori di implementazione: non è semplice inventare protocolli di sicurezza privi di vulnerabilità;
- interoperabilità: non è garantito che un protocollo sia implementato allo stesso modo da applicazioni diverse.

7.2.2 Sicurezza esterna delle applicazioni

Le applicazioni condividono anche la parte di sicurezza: tra il livello TCP e il livello applicazione è presente un livello di **sessione sicura** comune che rende sicuro il canale logico di rete, indipendentemente dall'applicazione che ha chiesto di crearlo.



Il livello ideale nella pila OSI sarebbe il livello di sessione (livello 5), in quanto una sessione non è altro che un canale logico, ma purtroppo questo livello nella pila TCP/IP è inglobato nel livello applicativo ⇒ il livello di sessione sicura è un livello di astrazione costituito dal codice di una libreria condivisa che estende l'interfaccia socket.

Esistono numerosi protocolli in competizione per creare canali sicuri:

- SSL/TLS (sez. 7.3): è il protocollo più usato al mondo;
- SSH: fu un prodotto di successo nel periodo di restrizione dell'esportazione di materiale crittografico dagli USA, ma oggi è per lo più una soluzione di nicchia;
- PCT: è stato proposto da Microsoft come alternativa a SSL, ma è stata un totale fiasco (da tempo è stato anche abbandonato da Microsoft).

Vantaggi

- semplicità: lo sviluppatore deve solo dichiarare di voler aprire un canale socket sicuro, specificando i parametri di sicurezza desiderati;
- errori di implementazione: lo sviluppatore non deve svilupparsi da sé la libreria di sicurezza;
- a scelta dell'applicazione: lo sviluppatore non è obbligato a usare l'interfaccia socket estesa, ma può usare un socket normale e implementare la propria libreria di sicurezza.

Svantaggio sicurezza di canale: la libreria apre un canale logico sicuro per trasmettere e ricevere sequenze di bit

7.3 SSL

Secure Socket Layer, proposto inizialmente da Netscape, è un protocollo per creare canali di trasporto sicuri.

SSL giace approssimativamente al livello sessione, tra il livello di trasporto e il livello applicativo:

- può essere usato con qualsiasi protocollo di livello applicazione (ad es. HTTP, SMTP, NNTP, FTP, TELNET);
- richiede un protocollo di trasporto affidabile che fornisca pacchetti non fuori sequenza e non mancanti al livello SSL (ad es. TCP).

7.3.1 Funzioni di sicurezza

- **autenticazione del server** (obbligatoria): all'apertura del canale, il server:
  1. si autentica inviando la propria chiave pubblica, tipicamente contenuta dentro un certificato X.509;
  2. subisce una sorta di sfida asimmetrica per dimostrare di conoscere la chiave privata corrispondente alla chiave pubblica nel certificato;
- **autenticazione del client** (facoltativa): all'apertura del canale, il client si autentica nello stesso modo del server;
- **autenticazione e integrità dei messaggi** (obbligatorie): tutti i dati trasmessi sono protetti da MAC keyed-digest (ad es. SHA-2) (la firma digitale sarebbe troppo lenta per flussi di dati);
- **riservatezza dei messaggi** (obbligatoria in SSL 2, facoltativa in SSL 3): tutti i dati trasmessi sono criptati con una chiave di sessione simmetrica (ad es. 3DES):
  - il client decide la chiave;
  - il client comunica la chiave al server tramite crittografia a chiave pubblica (ad es. RSA, DH);[2]
- **protezione da attacchi replay e filtering**: ogni messaggio viene numerato con un MID[3], garantendo una protezione completa possibile grazie al protocollo di trasporto affidabile (non parziale come in IPsec[4]):
  - replay: i messaggi devono arrivare sempre nello stesso ordine con cui sono stati inviati e non devono essere duplicati;
  - filtering (= cancellazione): non devono mancare dei messaggi.

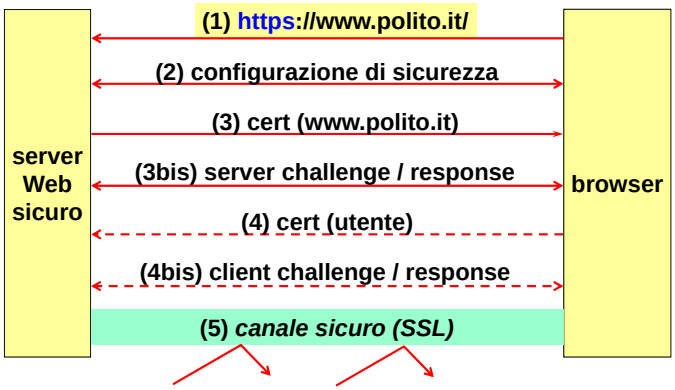
Alcune funzioni di sicurezza sono obbligatorie e altre sono facoltative:

- autenticazione delle controparti:
  - autenticazione del server: è obbligatoria perché è fondamentale difendersi contro gli attacchi shadow server e man-in-the-middle quando l'utente effettua acquisti su Internet;
  - autenticazione del client: è facoltativa perché:
    - può essere fatta anche a livello applicativo (ad es. nome utente e password);
    - l'utente di solito non ha un suo certificato;
    - nel commercio elettronico è importante che l'utente paghi, non la sua identità;
- protezione dei messaggi: secondo vari studi il 10% dei messaggi necessita realmente di riservatezza (facoltativa in SSL 3), mentre il 100% necessita di autenticazione e integrità (obbligatorie).

7.3.2 Schema concettuale

1. L'utente tramite un browser si collega ad un server sicuro.
2. A questo punto browser e server devono negoziare una configurazione di sicurezza, proprio perché è possibile andare ad utilizzare molti algoritmi sia lato server sia lato client. Il risultato di questa fase potrebbe anche essere che server e client non supportino lo stesso tipo di protocolli, e quindi non è possibile instaurare una connessione sicura.

[2] Si veda la sezione 2.3.1.
[3] Si veda la sezione 2.9.3.
[4] Si veda la sezione 5.9.9.

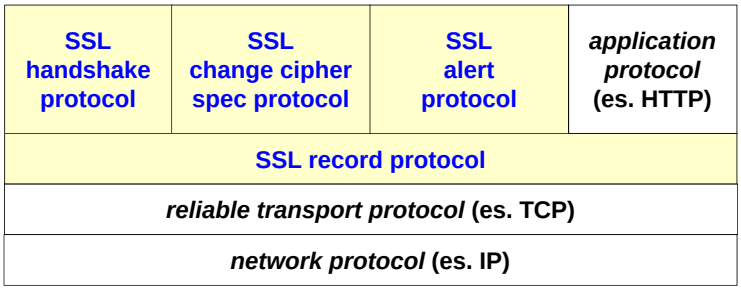


3. Ipotizzando di aver trovato una serie di parametri comuni, al passo successivo il server invierà al browser il suo certificato. Occorre notare che il certificato deve corrispondere esattamente con la URL a cui l'utente sta cercando di collegarsi, in moda da evitare shadow server o fake server.
   Ricevuta la chiave pubblica il browser esegue una serie di operazioni equivalenti ad uno scambio di challenge/response.

4. Opzionalmente il server può richiedere l'autenticazione del client, e in questo caso il browser provvederà ad inviargli il certificato dell'utente.
   Anche in questo caso questa fase è seguita da una serie di operazioni equivalenti ad un challenge/response in cui il browser deve dimostrare di controllare la chiave privata corrispondente alla chiave pubblica contenuta nel certificato.

Una volta che tutte queste fasi sono state eseguite correttamente si avrà un canale sicuro su cui sarà possibile trasferire tutti i dati desiderati, e che fornisce protezione verso possibili attacchi dall'esterno.

7.3.3 Architettura di SSL-3



Da un punto di vista architetturale SSL-3 si basa su un protocollo di rete (es. IP) e per funzionare correttamente ha bisogno di un protocollo di trasporto affidabile (e quindi questo significa TCP, ma non si è legati ad usare questo protocollo).

Su TCP si va a creare un *SSL record protocol*, ossia un protocollo per trasmettere singoli record SSL. Questi record possono essere utilizzati direttamente dai protocolli applicativi; ad esempio una trasmissione HTTP prende un payload HTTP e lo inserisce dentro un record SSL, il quale finirà in un segmento TCP che infine finirà in una serie di pacchetti IP.

L'SSL record protocol viene anche usato in fase di apertura del canale per fare il protocollo di handshake, ossia la fase di scambio iniziale durante la quale il server e il client si autenticano e negoziano le condizioni di sicurezza.

Nel caso in cui il canale rimanesse aperto per molto tempo è buona norma aggiornare periodicamente gli algoritmi e le chiavi associate. A tale scopo si ha un protocollo specifico chiamato *SSL change cipher spec protocol* che permette di cambiare le specifiche degli algoritmi e/o chiavi.

Infine l'*SSL alert protocol* lancia degli allarmi nel caso in cui, ad esempio, un pacchetto fosse duplicato, il MAC fosse errato e così via. Ovviamente anche questi tipi di messaggi devono a loro volta essere protetti, e quindi anche loro sono incapsulati in un SSL record protocol il quale fornisce le protezioni di base.

7.3.4 Session-ID

Si consideri la tipica transazione web, in cui un utente si collega ad una pagina, andando ad aprire un normale canale HTTP. Le operazioni che vengono svolte sono le seguenti:

1. open
2. GET page.htm
3. page.htm
4. close

A questo punto il browser inizia ad interpretare la pagina, e vede che la pagina richiede il caricamento di altre risorse. Di conseguenza sarà necessario andare nuovamente ad aprire il canale con il server:

1. open
2. GET home.gif
3. home.gif
4. close

1. open
2. GET home.mid
3. music.mid
4. close

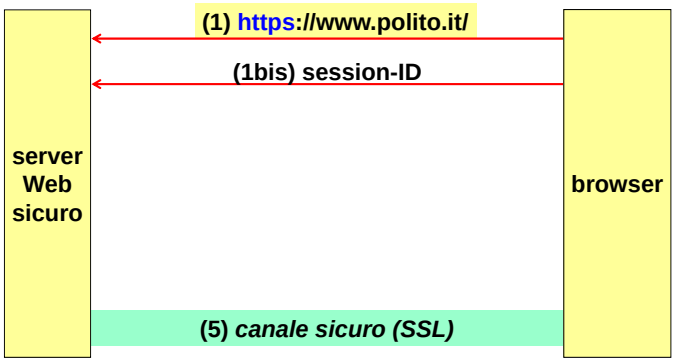
Se invece di usare un canale HTTP normale si usasse un canale sicuro SSL, questo metodo sarebbe ingestibile perché se ogni volta che bisogna scaricare una risorsa si deve andare a rinegoziare i parametri crittografici per SSL, il collegamento si appesantirebbe moltissimo.

Per evitare di ri-negoziare ad ogni connessione i parametri crittografici, il server SSL può offrire un session identifier, il cosiddetto *session-id*. Questo significa che più connessioni possono far parte di una stessa sessione logica (da qui il fatto che SSL implementa un livello sessione sicura più che singoli canali sicuri).

Se il client all'apertura della connessione presenta un session-id valido, si salta la fase di negoziazione e si procede subito con il dialogo SSL, andando quindi a saltare tutta la fase di handshake.

Si noti che l'uso dei session-id non è obbligatorio. Il server può anche rifiutare di utilizzarli (in assoluto, oppure dopo un certo periodo di tempo dalla loro emissione).

SSL con session-id



L'utente vuole collegarsi ad un server, ed insieme alla richiesta di connessione invia al server anche il session-id che gli è stato assegnato in una connessione precedente.

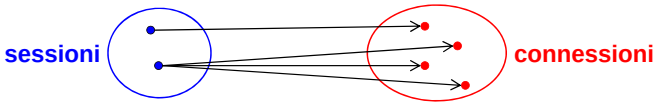
Il server verifica che questo session-id sia corretto, e in caso positivo apre direttamente il canale sicuro sul quale si può immediatamente iniziare a trasferire dati.

Il server si accorge se colui che sta cercando di usare una certa session-id ne sia anche l'effettivo proprietario, in quanto a tale identifier saranno associati una serie di parametri di sicurezza. Se un utente quindi cerca di usare un session-id non di sua appartenenza non saprà quali parametri andare ad utilizzare, e quindi il server non andrà ad aprire il canale sicuro.

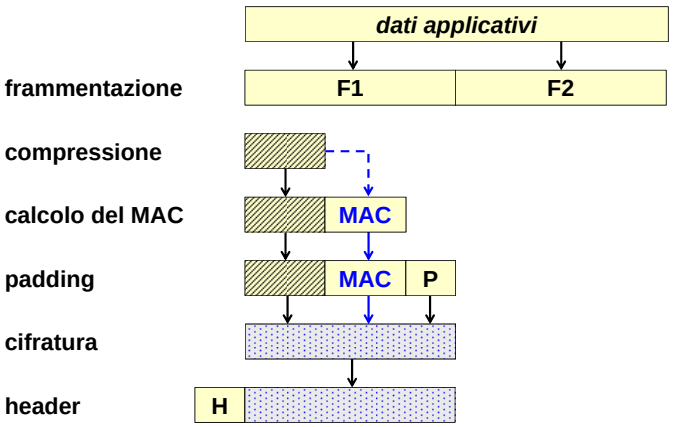
SSL/TLS: Sessioni e connessioni

Una *sessione SSL* è un'associazione logica tra client e server. Viene creata dal protocollo di handshake e definisce un insieme di parametri crittografici. Una sessione è usata come minimo da un collegamento o connessione SSL, ma potenzialmente da molte (nel caso in cui si fosse attivato il session-id).

Una *connessione SSL* invece è un canale SSL temporaneo tra client e server, corrispondente quindi ad un canale TCP, ed è associata ad una specifica sessione SSL. Si ha quindi una corrispondenza 1:1.



7.3.5 SSL-3 - TLS record protocol



Si supponga di avere dei dati applicativi che devono essere trasportati all'interno di un canale SSL.

1. Il primo passo che SSL esegue è quello di dividere i dati in più frammenti, anche se questi dati fanno parte di un qualche stream.

2. Prima di applicare a ciascuno di questi frammenti le proprietà di sicurezza tipiche del protocollo si effettua un'operazione (opzionale) di compressione. Questo processo riduce la quantità di dati che dovranno essere trasferiti ma richiede una potenza computazionale maggiore sia sul client che sul server. Nel caso della sicurezza però la compressione ha anche un altro grosso vantaggio: siccome la compressione tende ad eliminare le parti duplicate sostituendole con degli alfabeti comuni, questo permette di ridurre le informazioni disponibili per un eventuale attaccante che dovesse cercare di attaccare il cipher text. In generale quindi è sempre un'ottima idea fare la compressione dei dati prima di effettuarne la cifratura; fare la compressione dopo la cifratura non solo è inutile ma sarà addirittura dannoso in termini di calcoli.

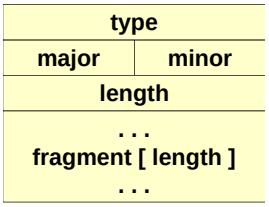
3. Successivamente il frammento (originale o compresso) viene utilizzato per il calcolo del MAC. In questo modo si fornisce integrità e autenticazione al singolo frammento.

4. Nell'ipotesi che venga utilizzato un algoritmo a blocchi, è necessario fare del padding in modo da far raggiungere al frammento una dimensione multiplo del blocco base. Questa operazione non è ovviamente necessaria nel caso in cui si fosse deciso di effettuare una cifratura di tipo stream.

5. A questo punto il frammento è pronto ad essere cifrato con l'algoritmo scelto, andando così a fornirgli anche riservatezza.

6. Infine al frammento cifrato viene premesso un piccolo header SSL.

TLS-1.0 - Record format



- *Type*: permette di specificare quali sono i dati trasportati, che possono essere dei protocolli di livello superiore o semplicemente dei dati applicativi.

- *Major & Minor*: identificano il major e il minor number del TLS. Si noti che per TLS-1.0 non viene scritto major=1 e minor=0, ma viene scritto major=3 e minor=1; questo perché TLS viene visto come la continuazione di SSL, e TLS ha iniziato ad esistere a partire dalla versione 3.1 di SSL.

- *Length*: campo di 16 bit che identifica la lunghezza del frammento. Se questo valore è ≤ 214 indica che i record non sono compressi e viene mantenuta questa codifica per compatibilità con SSL-2; i record compressi hanno invece un valore $\leq 214 + 1024$.

TLS - Calcolo del MAC

$$\text{MAC} = \text{message\_digest} (\text{key}, \text{seq\_number} \,||\, \text{type} \,||\, \text{length} \,||\, \text{fragment})$$

Per quanto riguarda il calcolo del MAC, esso viene calcolo con un message_digest che dipende dall'algoritmo che è stato negoziato tra il client ed il server in sede di handshake.

Il digest viene calcolato a partire dalla chiave di comunicazione condivisa; tale chiave sarà diversa a seconda che la comunicazione vada dal server verso il client o dal client verso il server. Tali chiavi sono note ad entrambi i peer ma non sono interscambiabili.

Il digest viene calcolato non soltanto sui dati del frammento vero e proprio, ma anche sulla sua lunghezza (per evitare che qualcuno effettui degli attachi di estensione), la versione del protocollo (per evitare attacchi che cercano di trasformare un pacchetto in un pacchetto di un'altra versione), il tipo (per proteggere l'integrità del frammento trasmesso) e anche il sequnce number (per proteggere da attacchi replay).

Si noti che il sequence number, un intero da 64 bit, non è scritto esplicitamente da nessuna parte; il suo valore è noto implicitamente da entrambi i peer in quanto essi contano tutti i frammenti.

In generale 64 bit sono sufficienti per proteggersi da qualunque tipo di attacco. Questo non è vero nel caso in cui SSL venisse usato per creare dei tunnel che rimangono attivi per molto tempo; in questo caso se si andasse a trasferire più di 2^{64} pacchetti si avrebbe un overflow, e per tale ragione per evitare di generarsi da soli un attacco di tipo replay bisognerebbe chiudere e riaprire il canale.

7.3.6 Problemi di SSL-2

Da qualche anno SSL-2 è stato deprecato in quanto era caratterizzato da una serie di problematiche di sicurezza:

1. SSL 2 era dotato di una versione esportazione, e in questa versione si aveva una riduzione a 40 bit di tutte le chiavi simmetriche, sia quella di cifratura sia quella usata per il calcolo del MAC. Il regolamento di esportazione prevedeva una riduzione a 40 bit solamente per le chiavi usate per la cifratura, ma non imponeva nessun vincolo per le chiavi di autenticazione; di conseguenza si aveva una riduzione senza motivo della forza del processo di autenticazione, e quindi non solo i dati erano spiabili ma erano anche falsificabili.

2. Il calcolo del MAC era fatto con un algoritmo di keyed-digest custom, che non era molto sicuro.

3. I byte di padding, usati per la cifratura nel caso di algoritmi a blocchi, sono inclusi nel calcolo del MAC ma la lunghezza del padding non lo è. Ciò permette per un attaccante attivo di cancellare byte alla fine dei messaggi.

4. *Cyphersuite rollback attack*: poiché la fase di handshake non era autenticata, un attaccante attivo poteva cambiare le cyphersuite nel messaggio di Hello in modo da forzare l'utilizzo di algoritmi di cifratura deboli. In ogni caso la cifratura doveva basarsi su chiavi di almeno 40 bit in quanto la cifratura è obbligatoria in SSL-2 (se il medesimo attacco fosse possibile anche in SSL-3 sarebbe un disastro in quanto in questa nuova versione è possibile togliere totalmente la cifratura).

7.3.7 SSL-3 - Soluzione ai problemi di SSL-2

- Le limitazioni riguardanti l'esportazione riguardano solamente le chiavi di cifratura, le quali sono ridotte a 40 bit. Le chiavi di autenticazioni sono invece lunghe tanto quanto è stato negoziato, tipicamente 128 bit.

- Per il calcolo del MAC si utilizza l'algoritmo HMAC, decisamente più forte rispetto al keyed digest custom usato in SSL-2.

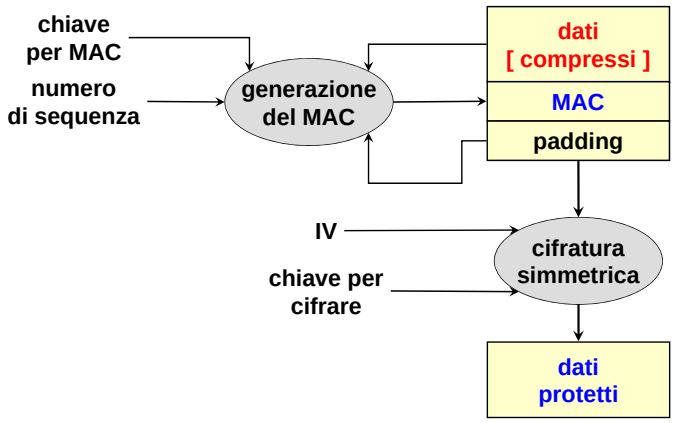
- Anche il padding è incluso nel calcolo del MAC.

- L'handshake è autenticato (ad eccezione dei messaggi di Change Cipher Spec) in modo indiretto tramite i messaggi di Finished. Quello che viene fatto è far eseguire completamente il processo di handshake, e una volta negoziati gli algoritmi e le chiavi nell'ultimo messaggio viene calcolato un valore che serve ad autenticare tutto lo scambio precedente. L'autenticazione quindi non riguarda i singoli messaggi ma l'intera procedura di handshake.

SSL-3 - Novità rispetto a SSL-2

- Compressione opzionale dei dati, da effettuarsi prima della cifratura.
- La cifratura dei dati è resa opzionale, in modo da avere solo autenticazione ed integrità.
- Aggiunta di un protocollo che fornisce la possibilità di rinegoziare la connessione. Questo permette di cambiare periodicamente le chiavi e gli algoritmi.

7.3.8 Protezione dei dati



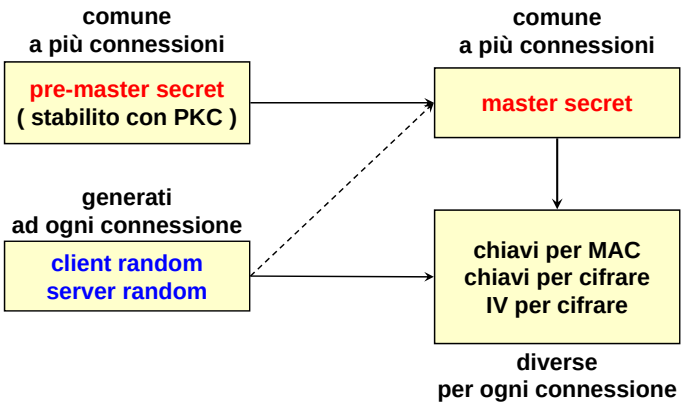
Per quanto riguarda la protezione dei dati, si vuole ora evidenziare quale relazione sussiste tra le varie chiavi che sono presenti.

Innanzitutto si hanno i dati, eventualmente compressi, che si desidera cifrare. Sapendo quanto sarà grande il MAC, è possibile andare a calcolare la dimensione del padding in modo automatico.

I dati e il padding, insieme alla chiave del MAC e al numero di sequenza, vengono forniti in input all'algoritmo che deve generare il MAC (negoziato durante l'handshake) il quale verrà inserito nel frammento.

A questo punto, supponendo di avere un algoritmo simmetrico a blocchi viene fatta la cifratura simmetrica del frammento tramite l'utilizzo di un vettore di inizializzazione (anch'esso implicito, entrambi i peer ne sono a conoscenza) e la chiave di cifratura.

7.3.9 Rapporto tra chiavi e sessioni



Innanzitutto la prima volta che due peer prendono contatto con SSL, stabiliscono tramite crittografia a chiave pubblica un cosiddetto *pre-master secret*; questo sarà comune a più connessioni.

A questo punto è possibile generare un *master secret*, anch'esso unico per la sessione e quindi comune a più connessioni.

Ogni volta che ci si collega invece, e quindi per ogni connessione, vengono generati due numeri random: *client random* e *server random*. Questi due numeri vengono usati insieme al pre-master secret nella prima connessione di una sessione per generare il master secret, ma soprattutto sono molto utili nelle volte successive perché vengono usate insieme al master secret per il calcolo tramite appositi algoritmi delle chiavi per il MAC, le chiavi per la cifratura simmetrica e il vettore di inizializzazione; tutti questi dati saranno quindi diversi per ogni connessione.

7.3.10 Meccanismi "effimeri"

Una delle particolarità di SSL rispetto ad altri protocolli è la possibilità di usare meccanismi cosiddetti *effimeri*. Questi sono rilevanti quando si trattano grosse quantità di dati o tanti collegamenti verso lo stesso server.

Un server SSL ha tipicamente una chiave pubblica per dimostrare la propria identità. Il pericolo nell'usare tante volte la medesima chiave pubblica è che a furia di fare firme l'attaccante possa desumere qualcosa circa la chiave privata corrispondente. Per evitare questa cosa il server può allora creare delle chiavi usa-e-getta asimmetriche generate al volo. Essendo però queste chiavi generate al volo, esse non saranno certificate. Quello che viene allora fatto è andare a firmare queste chiavi generate al volo utilizzando un certificato a chiave pubblica. Il server deve quindi essere dotato di un certificato non per usarlo direttamente nei confronti del client, ma per comunicare al client la chiave pubblica che è stata scelta per una certa connessione.

Nei meccanismi per lo scambio delle chiavi è possibile usare DH o RSA. La fase di generazione di una coppia di chiavi pubblica/privata è estremamente lenta se si usa RSA, ma molto veloce se invece si usa DH. Per tale ragione nel caso in cui venisse usato RSA, tipicamente la coppia di chiavi non viene usata per un'unica connessione ma viene riutilizzata più volte.

SSL permette inoltre di effettuare la cosiddetta *perfect forward secrecy*. Si supponga che ad un certo punto qualcuno riesca ad appropriarsi della chiave privata che è stata installata sul server. Se questa persona ha registrato tutte le precedenti sessioni, nel momento che lui scopre la chiave privata può decifrare non solo i pacchetti futuri ma anche tutti quelli passati.

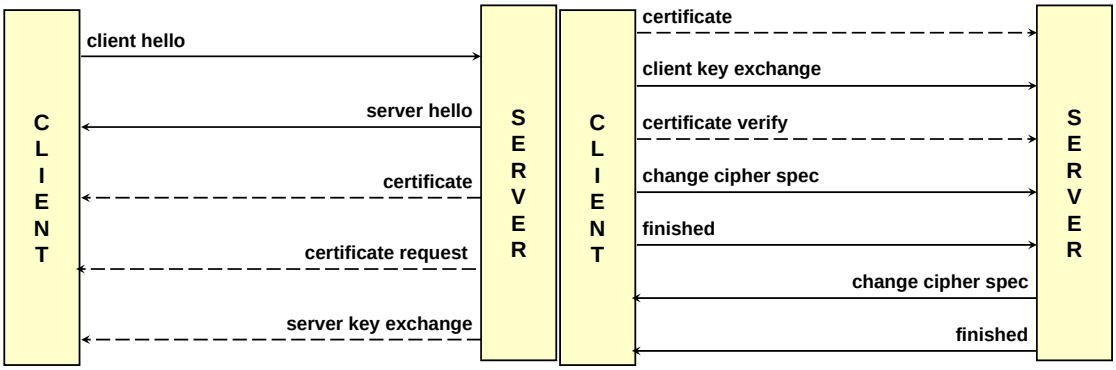
Se invece si è usato un meccanismo effimero, quello che viene fatto è ogni volta usare una coppia di chiavi pubblica/privata diversa per ogni connessione. Di conseguenza se anche qualcuno riuscisse ad attaccare la chiave privata del server, non potrebbe decifrare il traffico passato ma solo quello futuro, in quanto ogni connessione è stata protetta tramite una diversa chiave pubblica;

la chiave privata presente sul server non è usata per segretezza ma solo per firmare la chiave pubblica.

7.3.11 SSL-3 Handshake protocol

L'handshake protocol ha lo scopo di:

- Concordare una coppia di algoritmi che servono per confidenzialità ed integrità.
- Scambiare dei numeri casuali tra il client ed il server che servono come base per poi generare autonomamente le chiavi di crittografia.
- Stabilire una chiave simmetrica (da cui saranno poi derivate la chiave di cifratura e quella di autenticazione) tramite operazioni a chiave pubblica. Sono supportati RSA, DH e Fortezza.
- Negoziare il session-id.
- Scambiare tutti i certificati a chiave pubblica necessari.



In figura si hanno tutti i pacchetti che sono scambiati durante la fase di handshake. La linea tratteggiata indica che in una qualche implementazione del protocollo i pacchetti corrispondenti sono opzionali.

Il client inizia la comunicazione inviando un messaggio chiamato *client hello*, a cui il server risponde con un *server hello*.

Dopodichè sono opzionali, perché ad esempio si decide di usare il session-id, lo scambio del certificato del server (*certificate*), la richiesta del certificato al client (*certificate request*), un messaggio per lo scambio delle chiavi del serer (*server key exchange*), l'invio del certificato (*certificate*) nel caso in cui il client ne sia in possesso e il server ne abbia fatto richiesta.

È invece obbligatoria la *client key exchange*, che serve per generare le chiavi specifiche per questa connessione.

Opzionale ancora una fase di verifica del certificato (*certificate verify*).

Infine si ha l'invio del messaggio *change cipher spec*, che serve ad attivare i protocolli di protezione, e del messaggio *finished*, che serve a proteggere tutto lo scambio precedente. Quest'ultimi due messaggi sono anche inviati dal server verso il client per comunicare al client che anche lui sta per attivare tutte le misure di sicurezza e per proteggere i messaggi da lui inviati.

Client hello

Il client hello serve a:

- Dichiarare la versione di SSL preferita dal client.
- Inviare 28 byte generati in modo pseudo-casuali, il cosiddetto client random.

- Inviare un identificatore di sessione, il cosiddetto session-id. Questo campo avrà valore pari a 0 se il client desidera iniziare una nuova sessione, mentre avrà un valore diverso da 0 per chiedere di riesumare una sessione precedente. Non è detto che il server accetti valori diversi da 0, potrebbe infatti aver disabilitato il supporto per il session-id.
- Elencare tutte le cipher-suite del client. Con questo termine si intende l'insieme degli algoritmi di cifratura, di scambio chiavi e di integrità che il client è in grado di trattare.
- (A partire da SSL-3) Elencare i metodi di compressione supportati dal client.

Server hello

Il server hello viene utilizzato per inviare:

- La versione SSL scelta dal server.
- 28 byte pseudo-casuali, che costituiscono il server random.
- Un identificatore di sessione, il session-id. Se il client aveva inserito 0 come session-id, il server utilizzerà questo campo per inviare al client un nuovo session-id, altrimenti questo campo avrà valore uguale al session-id proposto dal client, ma solo se il server accetta di riesumare la sessione. In caso il server non accettasse, il server invierà al client un nuovo session-id.
- La cipher-suite scelta dal server; dovrebbe essere la più forte in comune con il client.
- Metodo di compressione scelto dal server, tra quelli proposti dal client.

Cipher suite

La cipher suite comprende:

- L'algoritmo di scambio chiavi.
- L'algoritmo di cifratura simmetrico.
- L'algoritmo di hash (per il MAC).

Questi vengono tipicamente indicati da delle stringhe, come ad esempio SSL_NULL_WITH_NULL_NULL (non viene usato nessun algoritmo per scambiare le chiavi, nessun algoritmo di cifratura simmetrico e nessun algoritmo di hash). Questa in particolare è la protezione che si ha inizialmente durante la fase di handshaking.

Certificate (Server)

Durante questa fase il server invia al client il certificato per la server authentication. Il certificato è di tipo X.509, e occorre che il contenuto del campo subject o del campo subjectAllName conincida con l'identità del server (e quindi deve contenere il nome DNS, l'indirizzo IP, ...).

Il certificato può avere attivo solo il keyUsage di firma o anche la parte di cifratura. Nel caso in cui il certificato è solo per firma allora è poi necessaria la fase di server-key exchange; questo serve ad indicare che il server userà meccanismi effimeri.

Certificate request

Messaggio inviato dal server al client, serve ad autenticare il client durante lo scambio SSL.

In particolare il server chiede al client di autenticarsi attraverso l'invio di un certificato X.509. Inoltre questo messaggio contiene anche l'elenco delle CA che il server ritiene fidate. Questo significa che il browser quando riceve questo messaggio guarda quali sono le CA che hanno rilasciato un certificato per l'utente, e ne cerca uno che sia stato emesso da una CA di cui il server si fida. Nel caso in cui l'utente non avesse un certificato emesso da una delle CA fidate, il protocollo si bloccherà in quanto il server non si fida dell'identità del client.

Server key exchange

Questa fase serve solamente nel caso in cui il server voglia usare dei meccanismi effimeri, e contiene la chiave pubblica per lo scambio chiavi del server.

In particolare questo messaggio è richiesto solo nei seguenti casi:

- Il server ha un certificato RSA solo per la firma.
- Si è scelto di usare DH anonimo o effimero per stabilire il master-secret. (Con DH anonimo si intende effettuare uno scambio chiavi DH in cui le chiavi non verranno firmate dal server. In questo modo il client non avrà nessuna certezza sull'identità del server, ma si avrà comunque protezione del canale di comunicazione da terze parti)
- Ci sono problemi di esportazioni e quindi si devono usare chiavi RSA/DH temporanee.
- Obbligatorio se si utilizza Fortezza.

Questo è l'unico messaggio che contiene una firma esplicita asimmetrica del server, proprio perché la chiave messa dentro questo scambio deve essere in qualche modo autenticata.

Certificate (client)

Nel caso in cui il server abbia richiesto un certificato al client e questo ne disponga, viene allora inviato il messaggio di certificate client.

Questo messaggio trasporta il certificato per la client authentication. Il certificato deve essere stato emesso da una delle CA fidate elencate dal server nel messaggio di certificate request.

Client key exchange

Il client fornisce le informazioni che servono per determinare le chiavi di cifratura e di MAC. Esistono varie possibilità:

- Il client può mandare il pre-master secret che lui si è inventato, cifrato con la chiave pubblica RSA del server (temporanea, nel caso si usassero meccanismi effimeri, o del certificato).
- Possibilità di inviare la parte pubblica di DH, nel caso in cui fosse stato deciso uno scambio chiavi basato su DH.
- Invio dei parametri dell'algoritmo Fortezza.

Certificate verify

La fase di certificate verify è una fase in cui viene fatta una prova esplicita di firma.

Viene calcolato l'hash su tutti i messaggi di handshake che precedono questo, e viene firmato con la chiave privata del client.

Questo è necessario solamente nel caso di client authentication, ossia solamente nel caso in cui il server avesse richiesto al client di autenticarsi. In un certo senso questa è la risposta alla sfida da parte del client verso il server.

Change cipher spec

Il change cipher spec è uno dei due messaggi finali. Dopo questo messaggio tutti i successivi frammenti vengono protetti con gli algoritmi e le chiavi che sono stati negoziati nei messaggi precedenti.

A rigore questo messaggio fa parte di un protocollo specifico e non dell'handshake.

Varie analisi indicano come sia eleminabile, ma per il momento non è stato eliminato né in SSL né nelle versioni TLS successive.

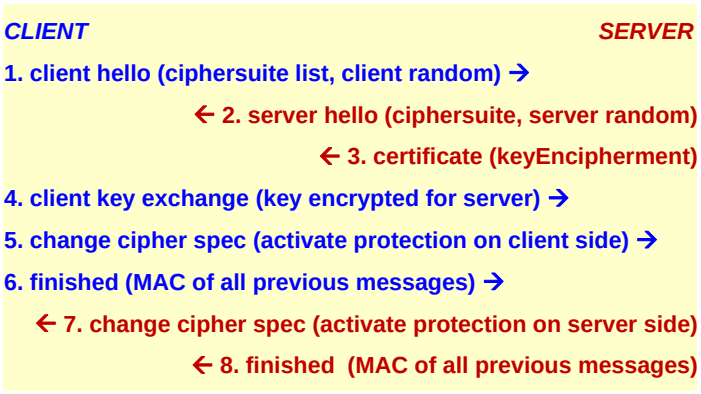
Finished

Questo è il primo messaggio che viene protetto con gli algoritmi negoziati, ed è fondamentale per autenticare tutto lo scambio perché:

- Contiene un MAC calcolato su tutti i messaggi di handshake precedenti (ad eccezione del change cipher spec) tramite l'uso del master secret.
- Evita attacchi di tipo man-in-the-middle rollback, atti a fare version downgrade o ciphersuite downgrade.
- È diferenziato per client e server perché ognuno dei due calcola il MAC sui messaggi che lui ha inviato.

7.3.12 TLS - Esempi di implementazione

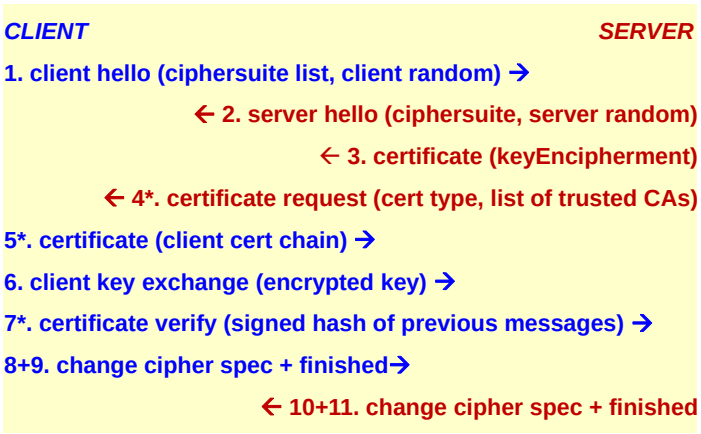
TLS, no chiave effimera, no client auth



Questo è lo schema di collegamento più semplice che ci sia, quello che minimizza il numero di messaggi scambiati.

1. Il client manda il *client hello*, in cui gli elementi fondamentali sono il client random e la ciphersuite list.
2. Il server risponde con un *server hello*, in cui ha scelto una specifica ciphersuite e invia il suo server random.
3. A questo punto il server manda il messaggio di *certificate*, in cui la cosa importante è che il certificato che ha mandato serve non solo per firma digitale ma anche per cifratura dei dati.
4. Il client, usando il client random e il server random, genera il master secret e invia questa chiave cifrata al server utilizzando il certificato inviatogli dal server nel mesaggio precedente.
5. Il client può a questo punto attivare tutte le misure di sicurezza lato client tramite l'invio del messaggio *change cipher spec*.
6. Invio del messaggio *finished*, che contiene il MAC di tutti i messaggi precedenti (ad eccezione del change cipher spec).
7. Il server risponde con il suo *change cipher spec*, con il quale afferma l'intenzione di attivare le misure di sicurezza lato server a partire dal prossimo messaggio.
8. Invio del messaggio di *finished* che contiene il MAC dei messaggi 2 e 3.

TLS, no chiave effimera, client auth

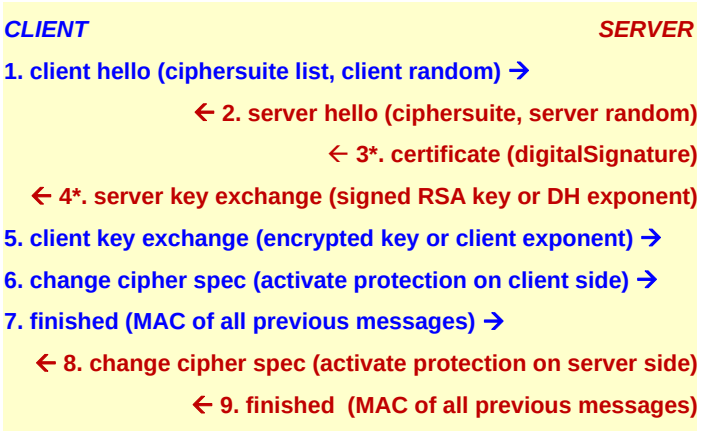


1. *Client hello.*
2. *Server hello.*
3. *Certificate (server).*
4. Il server invia al client un messsaggio di *certificate request*, contenente quale tipo di certificato il client dovrà avere e l'elenco delle CA di cui il server si fida.
5. Il client manda il proprio certificato sotto forma di client cert chain, perché il server ha indicato le root CA di cui lui si fida e di conseguenza il client deve mostrare come dalla CA che ha emesso il suo certificato si può arrivare ad una root CA fidata.
6. Inoltre il client invia un messaggio di *client key exchange*, con il quale il client invia la chiave cifrata con i meccanismi del server.
7. Fase di *certificate verify*, che è l'unico caso di firma digitale esplicito da parte del browser. Si effettua il calcolo dell'hash di tutti i messaggi precedenti, firmato con la chiave privata del client. Questo viene fatto per dimostrare di possedere la chiave privata corrispondente al certificato che è stato mandato al passo 5.

8+9. *Change cipher spec + finished.*

10+11. *Change cipher spec + finished.*

TLS, chiave effimera, no client auth

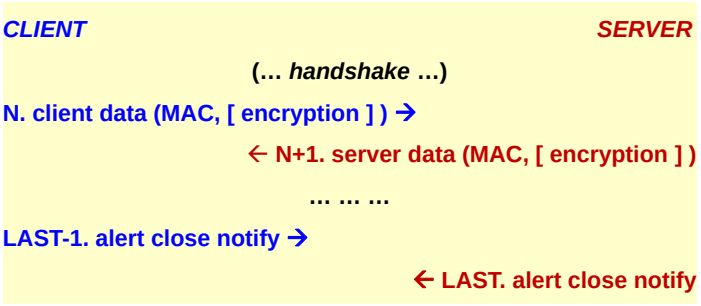


1. *Client hello.*
2. *Server hello.*
3. *Certificate (server)*, questa volta contenente la dicitura digitalSignature: questo significa che il certificato di cui il server dispone non può essere usato per fare cifratura, ma solo per fare firme digitali. Il client quindi non può inviare al server il segreto cifrato che serve per derivare le chiavi.
4. Come conseguenza il server sceglie di generare le chiavi, ad esempio una chiave RSA o un esponente DH, e manda la componente pubblica al client firmata tramite la chiave privata corrispondente al certificato mandano al passo 3.
5. Il client invia al server un numero random da lui generato, cifrato con la chiave inviatogli dal server al passo 4. Nel caso però il server avesse inviato un esponente DH, il client non manda un random ma manda anche lui un esponente DH ma senza firma. A questo punto c'è da chiedersi com'è possibile essere sicuri che quella sia la componente DH scelta dal client e non qualcosa manipolato un soggetto esterno... L'integrità dell'esponente è garantita dall'operazione eseguita al punto 7, in cui si va a calcolare il MAC su tutti i messaggi inviati dal client! Questo fornisce una firma implicita all'esponente.

6+7. *Change cipher spec + finished.*

8+9. *Change cipher spec + finished.*

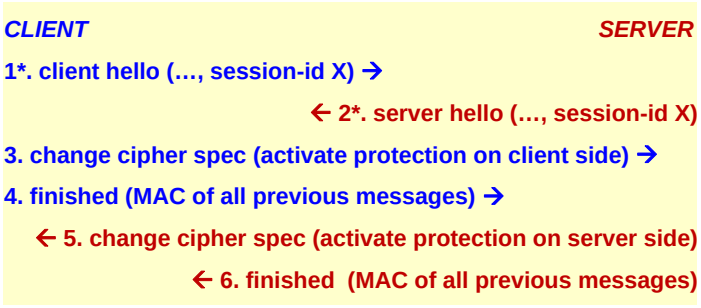
TLS, scambio dati e chiusura canale



1. Fase di handshake conclusa con successo.
2. Al messaggio N-esimo il client invia i suoi dati protetti con il MAC e opzionalmente con la cifratura.
3. Il server risponde con il messaggio $N+1$, contenente i suoi dati (anch'essi protetti con il MAC e opzionalmente cifrati).
4. Ad un certo punto si decide di chiudere il canale. Il messaggio LAST-1 è di tipo *alert close notify*, facente parte dell'alert protocol, che serve a notificare al server che il client sta per chiudere il canale.
5. Il server risponde al messaggio del client con un *alert close notify*, con il quale il server notifica che anche per lui il canale è stato chiuso.

Si noti che entrambi i messaggi 4 e 5 sono protetti con un MAC, perché altrimenti un terzo potrebbe chiudere il canale contro la volontà dei due peer.

TLS, ripresa di una sessione



1. Nella fase di *client hello* tra i vari parametri viene indicato un session-id X.
2. Nel *server hello*, nel caso in cui il server accettasse di continuare una sessione precedente, sarà presente lo stesso valore per il session-id.
3. A questo punto tutte le altre fasi dell'handshake vengono saltate, in quanto il server è a conoscenza dei parametri precedentemente negoziati associati al session-id X. Si ha quindi l'invio del *change cipher spec*, che attiva la protezione lato client.
4. Messaggio di *finished*, contenente il MAC del messaggio 1.

5+6. *Change cipher spec* + *finished*.

7.3.13 Transport Layer Security (TLS)

A partire dalle versione 3.1 SSL ha cambiato nome in *Transport Layer Security (TLS)*. Questo è dovuto al fatto che Netscape, autore della versione inziale di SSL, nel momento in cui ha terminato la sua attività ha donato il protocollo all'IETF. Si è però avuta la fortissima opposzione di Microsoft che non poteva accettare che un protocollo inventato da un concorrente venisse standardizzato con un nome che richiamasse il concorrente stesso, alla fine l'IETF ha ceduto a tali pressioni e ha deciso di usare il nome TLS.

Il TLS-1.0 è documentato nell'RFC-2246 del gennaio 1999, e implementa praticamente SSL-3.1 (coincide al 99% con SSL-3). Le uniche modifiche che sono state fatte sono:

- Uso di algoritmi standard, come ad esempio HMAC invece del keyed-digest custom inventato da Netscape.
- Enfasi su algoritmi che non richiedessero pagamenti di royalties. La ciphersuite base utilizza DH + DSA + 3DES, ossia TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA.

TLS 1.1 (SSL 3.2)

La versione successiva del protocollo porta alcuni miglioramenti, in particolare aiuta a proteggersi contro una serie di attacchi verso CBC. Nella prima versione di TLS era possibile creare dei cosiddetti oracoli crittografici nel caso in cui veniva usato CBC con un padding errato, ossia andando a cambiare i bit di padding e fornendoli al server l'attaccante può usare il server per cercare di capire se ha sbagliato o non ha sbagliato a fare le modifiche; questo attaco si basa sul fatto che il server fornisce delle risposte diverse a seconda che il MAC sia sbagliato o la decifratura sia sbagliata.

Per evitare questo genere di attacchi il vettore di inizializzazione implicito che veniva calcolato e mai trasmesso viene ora sostituito con un vettore di inizializzazione esplicito. Gli eventuali errori nel padding usano il messaggio generico di alert bad_record_mac, che non distingue più tra errori nel padding e errori sui dati; nella versione precedente di TLS si usava il messaggio

decryption_failed, e quindi si andava a fornire delle informazioni all'attaccante se l'attacco era stato fatto sulla parte di padding o sulla parte di dati.

Minori miglioramenti comprendono:

- Assegnazione alla IANA del compito di registrare tutti i parametri del protocollo.
- Il fatto che un canale venga chiuso prematuramente non implica più che non si possa riprendere la sessione.
- Assegnazione agli sviluppatori di una serie di note addizionali per chiarire vari possibili attacchi.

TLS 1.2 (SSL 3.3)

Versione standardizzata nell'agosto del 2008 con l'RFC-5246.

Principali modifiche:

- La ciphersuite è stata estesa per specificare anche le PRF (pseudo-random function), le funzioni che il client ed il server usano per generare i numeri random da utilizzarsi per la generazione delle chiavi.
- Visto che SHA-1 era ormai stato deprecato, inizia ad essere usato SHA-256 nei messaggi di Finished (in modo che sia forte la parte di autenticazione dell'handshake) e viene reso opzionale (ma molto consigliato) di usarlo anche nella parte di MAC.
- Introduzione del supporto per authenticated encryption basato sull'uso di AES in modalità GCM o CCM.
- Incorpora le estensioni (meccanismi per aggiungere altre parti al protocollo) e tutte le ciphersuite basate su AES che prima erano descritte in un RFC a parte.
- La default ciphersuite diventa TLS_RSA_WITH_AES_128_CBC_SHA.
- Diventano deprecate le ciphersuite basate su IDEA e DES.

7.3.14 TLS e server virtuali

All'interno di HTTP/1.1 quando un client si collega ad un server oltre ad indicare nella GET o nella POST la risorsa a cui si desidera accedere, è anche obbligatorio l'uso di un header Host che indica l'host virtuale a cui ci si vuole collegare. Questo viene fatto perché ad un unico indirizzo IP (e quindi una macchina fisica) possono corrispondere più server web virtuali.

Questa distinzione è fatta facilmente in HTTP, ma è difficile da fare con HTTPS perché TLS viene attivato prima di HTTP, ma il server deve inviare al client il certificato corrispondente al nome del server a cui il client vorrà collegarsi. Tuttavia il client fornirà questa informazione solamente all'interno del protocollo HTTP, e di conseguenza il server avendo al suo interno più certificati non saprà quale di questi dover inviare al client.

Per risolvere questo problema è possibile:

- Usare dei certificati collettivi (wildcard) nel caso in cui i vari server virtuali ospitati corrispondano tutti al medesimo dominio. Si ha quindi un'unica chiave privata condivisa tra tutti i server.
  Tuttavia questa soluzione ha il problema che i certificati collettivi vengono trattati in modo diverso dai vari browser.
- Creare un certificato per il server che contenga un campo vuoto nella parte di subject e contenga invece come subjectAltName tutti i nomi dei vari server virtuali ospitati. Anche in questo caso la chiave privata è condivisa tra tutti i server virtuali.
  Con questa soluzione si ha il problema di dover ri-emettere il certificato ad ogni aggiunta o cancellazione di un server virtuale.

- Usare l'estensione *Server Name Indication (SNI)*, documentata nell'RFC-4366. In particolare durante la fase di ClientHello il client può mandare la SNI, contenente il nome del server virtuale a cui il client vorrà collegarsi.
  Il problema di questa soluzione è che trattandosi di una estensione il suo supporto non è obbligatorio, e quindi non è utilizzabile da tutti i browser e server.

7.3.15 Datagram Transport Layer Security (DTLS)

Dato il grande successo di TLS è stato proposto il protocollo DTLS, che altro non è che l'applicazione dei concetti di TLS al trasporto di datagrammi (e quindi potenzialmente fornire protezione ad UDP).

Tuttavia visto che parte delle protezioni offerte da TLS sono possibili grazie al fatto che il suo uso è associato all'uso di un protocollo di trasporto affidabile, essendo UDP un protocollo di trasporto non affidabile non sarà possibile avere tutti i servizi di sicurezza offerti.

Per tale ragione DTLS non è supportato da tutti, o comunque è in competizione con altre soluzioni. Ad esempio è in competizione con IPsec e con la possibilità di implementare la sicurezza direttamente a livello applicativo.

Ad esempio, nel caso di protezione del protocollo SIP è possibile usare:

- IPsec, e si ha quindi il problema di configurare correttamente il client ed il server.
- TLS, nel caso in cui si decidesse di usare SIP_over_TCP.
- DTLS, nel caso si usasse SIP_over_UDP.
- Secure SIP, in cui la sicurezza è implementata direttamente all'interno del protocollo SIP.

7.3.16 Heartbleed

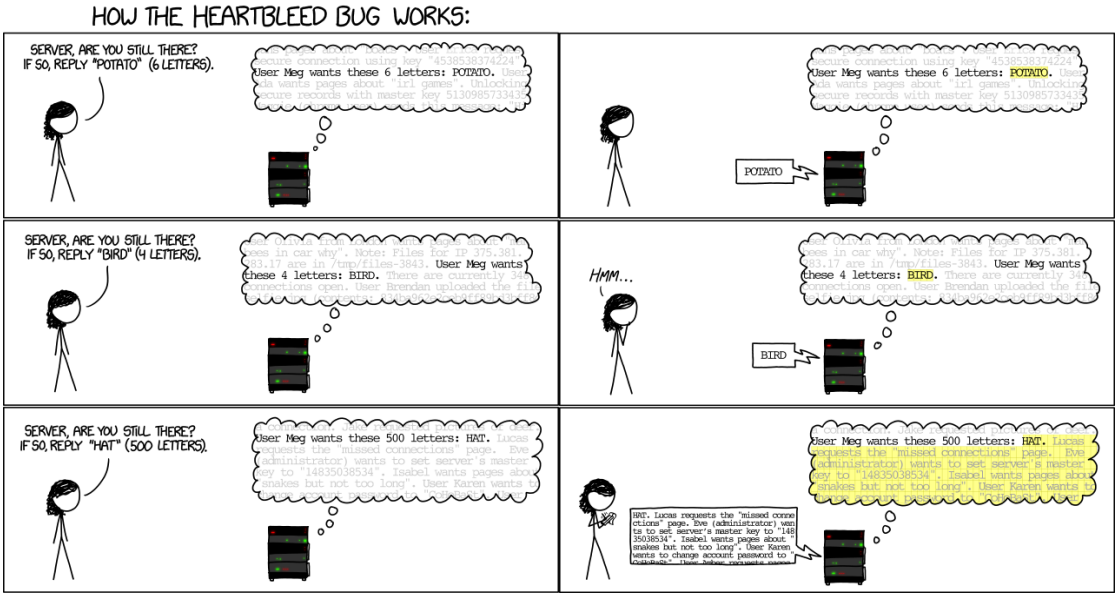
Heartbleed è un attacco scoperto nell'aprile del 2014.

Tutto parte dall'RFC-6520, che inserisce un'estensione chiamata di Heartbeat. Questo concetto è presente anche all'interno di altri protocolli di rete, e consiste nell'inviare periodicamente dei pacchetti tra due peer quando questi non stanno scambiando nessun dato, con il solo scopo di non far terminare la connessione. In TLS questa estensione è usata per tenere una connessione aperta in modo da non dover continuamente rinegoziare i parametri. Questa è molto importante per SSL, ma è assolutamente vitale per DTLS perché in questo caso non c'è il concetto di sessione, e quindi si corre il rischio di dover ogni volta rinegoziare dall'inizio tutti i parametri.

Inoltre questa estensione è anche utile in PMTU discovery, ossia vengono mandati dei messaggi di heartbeat per scoprire qual è la path MTU migliore tra il client ed il server.

Il problema di sicurezza legato a questa estensione è documentato nel CVE-2014-0160 (CVE = Common Vulnerability Exposion, che è un DB mondiale che indica tutte le vulnerabilità note. Ad ogni nuova vulnerabilità è associato un numero composto dall'anno e dal numero all'interno di quell'anno), e non riguarda in generale il protocollo TLS ma riguarda la specifica implementazione all'interno di openssl. In particolare si ha un problema di buffer over-read, ossia il server TLS risponde con più dati (sino a 64 KB) di quanti inviati nella richiesta di heartbeat.

Come conseguenza si ha che l'attaccante può ottenere dati sensibili presenti in quel blocco di RAM, quali user+pwd o addirittura la chiave privata del server nel caso non venisse usato un HSM.



7.4 Sicurezza in HTTP

SSL è il principale protocollo di sicurezza utilizzato in Internet, ma non è l'unico. Esistono infatti altri protocolli che si applicano soprattutto quando si vanno ad utilizzare delle applicazioni Web.

In generale con il termine applicazioni Web si intende applicazioni che trasportano i loro dati tramite il protocollo di livello 7 HTTP.

Correntemente esistono due versioni in uso del protocollo HTTP, ossia HTTP/1.0 e HTTP/1.1. All'interno dell'RFC che definisce la prima versione del protocollo sono definiti due meccanismi di sicurezza molto basilari:

- *Address-based*: il server HTTP controlla l'accesso alle sue risorse in base all'indirizzo IP del client.
- *Password-based* (Basic Authentication Scheme): l'accesso ad una determina risorsa è limitato da una username e password, che vengono trasmesse codificate in Base64. Tuttavia questo non è un algoritmo di cifratura, e quindi serve solo ad eseguire una sorta di offuscamento dei dati.

Entrambi gli schemi risultano essere altamente insicuri, e questo è già riconosciuto dagli autori dell'RFC che hanno scritto che HTTP suppone che i suoi meccanismi siano usati all'interno di un canale sicuro.

HTTP/1.1 introduce un nuovo meccanismo di sicurezza basato su sfida simmetrica chiamato *digest authentication*.

Attualmente tutti questi meccanismi di sicurezza sono riassunti nell'RFC-2617 che parla di autenticazione a livello HTTP.

7.4.1 HTTP - Basic authentication scheme

```
GET /path/alla/pagina/protetta HTTP/1.0
  HTTP/1.0 401 Unauthorized - authentication failed
  WWW-Authenticate: Basic realm="RealmName"
Authorization: Basic B64_encoded_username_password
  HTTP/1.0 200 OK
  Server: NCSA/1.3
  Content-type: text/html
  <HTML> pagina protetta ... </HTML>
```

Dopo aver aperto un canale HTTP, il client chiede una risorsa utilizzando ad esempio la funzione GET. Da un punto di vista del client le pagine protette sono del tutto indistinguibili da quelle non protette (si guardi il path di esempio della risorsa, per il client questa non è per niente significativa).

Come risposta a questa richiesta il client riceverà un messaggio di errore HTTP/1.0 401, accesso non autorizzato; questo errore è di livello 4, ossia è un errore temporaneo. Esso è classificato in questo modo perché il client non aveva modo di sapere che era necessario autenticarsi per avere accesso a tale risorsa. Come conseguenza il canale HTTP non viene chiuso, ma viene lasciato aperto nel caso in cui il client volesse autenticarsi.

A tale scopo il server comunica al client tramite l'header WWW-Authenticate che per avere accesso alla risorsa è richiesta un'autenticazione basic per un certo reame. Si noti che questo reame non è da confondere con il reame di Kerberos. Questo è solo un suggerimento che il server dà al client per permettergli di ricordarsi quale username e password sono da usare. Si potrebbe quindi avere realm = "Portale della didattica del Politecnico di Torino".

Quando il browser riceve questo header mostra tipicamente un popup automatico (non scritto dal programmatore della pagina web) in cui è scritto qualcosa del tipo "Il server ha richiesto autenticazione per il reame..." e chiede di inserire username e password.

L'utente andrà allora ad inserire username e password in questo popup, e sarà poi compito del browser di andare a trasmetterli nel canale HTTP rimasto aperto tramite l'uso dell'header Authorization. In questo header si ha l'indicazione che si sta cercando di effettuare un'autorizzazione basic, seguita dall'username e dalla password codificati in Base64. Il livello di sicurezza di questo meccanismo è molto basso, perché chunque intercetti questo header può tranquillamente leggere username e password semplicemente decodificando i dati.

Supponendo che nessuno intercetti questa comunicazione, il server riceverà l'header Authorization e ne andrà a fare la decodifica. Se user/pwd sono validi e soprattutto sono autorizzati ad avere accesso alla risorsa (autenticazione e autorizzazione vengono effettuate in un unico passo), allora il server risponderà alla richiesta originale con un messaggio HTTP/1.0 200 a cui seguirà la risorsa richiesta.

7.4.2 HTTP - Digest authentication

La digest authentication è un miglioramento del meccanismo basic, che permette di fare un'autenticazione sicura anche al di fuori di un canale SSL.

Questo meccanismo è stato introdotto nell'RFC-2069, ormai tecnicamente obsoleto, ma viene considerato come caso base nell'RFC-2617.

Quello che in pratica viene fatto è calcolare un keyed-digest basato sulla password dell'utente:

$$\begin{gathered} \text{HA1} = \text{md5}\ (\ \text{A1}\ ) = \text{md5}\ (\ \textit{user}\ \text{":"}\ \textit{realm}\ \text{":"}\ \textit{pwd}\ ) \\ \text{HA2} = \text{md5}\ (\ \text{A2}\ ) = \text{md5}\ (\ \textit{method}\ \text{":"}\ \textit{URI}\ ) \\ \text{response} = \text{md5}\ (\ \text{HA1}\ \text{":"}\ \text{nonce}\ \text{":"}\ \text{HA2}\ ) \end{gathered}$$

Il nonce, inviato dal server al client, è utilizzato allo scopo di evitare attacchi di tipo replay.

Il server di autenticazione può anche inserire un campo "opaque", ossia una stringa binaria non intelligibile dal browser, per trasportare eventuali informazioni di stato (es. token SAML).

Digest authentication scheme

```
GET /private/index.html HTTP/1.1
  HTTP/1.0 401 Unauthorized - authentication failed
  WWW-Authenticate: Digest realm="POLITO",
    nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
    opaque="5ccc069c403ebaf9f0171e9517f40e41"
Authorization: Digest username="lioy",
  realm="POLITO",
  nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
  uri="/private/index.html",
  response="32a8177578c39b4a5080607453865edf",
  opaque="5ccc069c403ebaf9f0171e9517f40e41"
  HTTP/1.1 200 OK
  Server: NCSA/1.3
  Content-type: text/html
  <HTML> pagina protetta ... </HTML>
```

pwd=antonio

Lo schema inizia con una GET da parte del client di una pagina protetta ben precisa. Questa richiesta genera il messaggio di errore HTTP/1.1 401, seguito dall'header WWW-Athenticate con il quale il server indica che occorre effettuare un'autenticazione di tipo Digest, per il reame = "Polito". All'interno dell'header sono anche presenti il nonce, che verrà usato per il calcolo del digest, e l'opaque, che il client dovrà rinviarlo inalterato.

Al ricevimento di questo messaggio il browser chiede all'utente di inserire username e password, e in base a queste credenziali andrà a costruire la risposta, che verrà inviata al server tramite l'header Authorization. Se il digest calcolato sarà corretto e l'opaque sarà uguale all'originale, il server risponderà in modo affermativo con HTTP/1.1 200 allegando la risorsa richiesta.

7.4.3 HTTP e SSL/TLS

HTTP viene spesso usato insieme a SSL/TLS. Si hanno due alternative:

- *TLS then HTTP*, ovvero prima viene aperto un canale sicuro e sucessivamente viene utilizzato HTTP per il trasporto dei dati.
  (SSL then HTTP è usato ma non è stato mai documentato perché inizialmente SSL era un protocollo proprietario di Netscape)
- *HTTP then TLS*. In questo caso si ha l'apertura iniziale di un normale canale HTTP, e sarà il server a decidere se si vuole fare sicurezza oppure no. Si ha quindi la possibilità di fare upgrading verso TLS di un normale canale HTTP/1.1.

Le due alternative non sono equivalenti, soprattutto perché hanno un impatto diverso sulle applicazioni, sui firewall e sugli IDS.

- In un'applicazione basata su TLS then HTTP la sicurezza non è gestita dal programmatore del server Web, ma è gestita dal sistemista: è compito di quest'ultimo attivare TLS prima di rendere accessibile l'applicazione Web. Questo ha il vantaggio di richiedere un minore lavoro agli sviluppatori, ma allo stesso tempo essi non avranno nessun controllo sui sistemi di sicurezza; occorrerà fidarsi dell'operato dei sistemisti. Occorre quindi avere una sincronizzazione tra il lavoro dei programmatori e quello dei sistemisti.
  In un'applicazione basata invece su HTTP then TLS è lo sviluppatore che deve occuparsi della sicurezza: nel momento in cui un utente richiede di accedere a delle risorse protette occorre cessare il normale dialogo HTTP, trasformare il canale in sicuro e solo a quel punto potrà riprendere il dialogo HTTP. Questo si traduce in un carico di lavoro maggiore per lo sviluppatore in quanto dovrà occuparsi anche della sicurezza, ma ciò può anche essere visto

come un vantaggio in quanto è lo sviluppatore stesso che controlla se e quale sicurezza si desidera avere.

- Per quanto riguarda i firewall le due soluzioni non sono equivalenti principalmente per il motivo che esse utilizzano porte diverse. Se si utilizza TLS then HTTP si avrà la porta 80 per HTTP non sicuro, e la porta 443 per TLS su cui si va a veicolare HTTP. Di conseguenza il firewall potrà chiaramente distinguere HTTP sicuro da quello non sicuro.
  Con HTTP then TLS invece si ha solamente un canale, TCP 80, il quale a discrezione dello sviluppatore potrà essere reso sicuro per la trasmissione di determinate risorse. Di conseguenza il firewall, basandosi solo ed esclusivamente sulle porte, non riesce più ad effettuare una distinzione tra canali sicuri e quelli non sicuri. Sarebbe ad esempio necessario effettuare un content exception.

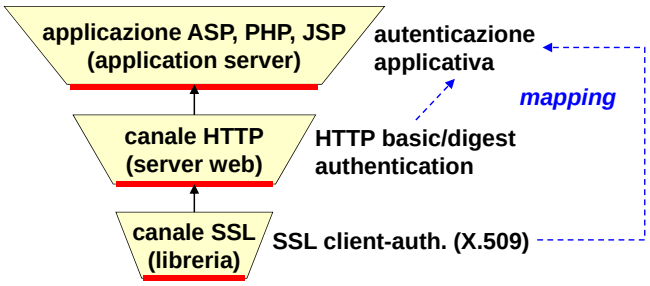
- Per quanto riguarda gli IDS, se si utilizza TLS then HTTP l'IDS sarà solamente in grado di vedere una richiesta di apertura di un canale protetto. Una volta che il canale sarà aperto non sarà più possibile vedere nulla.
  Con HTTP then TLS invece l'IDS può vedere che il canale è stato creato e che per una specifica richiesta il server ha deciso di trasformare il canale in sicuro. L'IDS non sarà comunque in grado di vedere nulla fintanto che TLS sarà attivo, ma almeno si ha un controllo su quali URL viene attivata la sicurezza.

Si noti che questi concetti non sono validi solamente se si utilizza HTTP, ma in generale sono validi per tutti i protocolli di livello 7 basati su TCP, e che quindi possono essere basati su TLS.

7.4.4 Client authentication SSL a livello applicativo

Tramite la client authentication è possibile identificare l'utente che ha aperto un canale senza andare a richiedergli username e password, ma basandosi sull'uso di un certificato.

Alcuni server Web permettono di fare un mapping (semi-)automatico tra le credenziali che sono state estratte dal certificato X.509 utilizzato a livello SSL e gli utenti del servizio HTTP e/o del SO.



L'autenticazione di un utente può avvenire in tre modalità diverse:

- Nel caso in cui il canale SSL è stato attivato con client authentication, per autenticarsi l'utente dovrà fornire un certificato X.509, oltre a dover rispondere correttamente alla sfida. In quasto caso il server può effettuare un mapping per far sapere allo sviluppatore applicativo le credenziali dell'utente che ha richiesto accesso ad una determinata risorsa.

- Se si decide di usare SSL solo con server authentication, il prossimo punto dove si ha la possibilità di identificare l'utente è a livello HTTP, dove è possibile richiedere una basic o una digest authentication. Anche in questo caso il risultato dell'autenticazione può essere mappato.

- Se non si decide di usare nessuna delle sue soluzioni precedenti, l'ultima opzione è che dentro la pagina Web, e quindi all'interno del codice, ci sia un form dentro cui si va a chiedere all'utente di inserire username e password.

Queste tre soluzioni sono sicuramente equivalenti per quanto riguarda il risultato finale, in quanto in ogni caso l'applicazione Web saprà l'identità dell'utente che ha aperto il canale, ma non sono per niente equivalenti dal punto di vista della sicurezza.

Qui entra in gioco un concetto oggigiorno molto importante chiamato *superficie di attacco*. Visto che il codice è il punto più debole di qualunque sistema, si deve cercare di esporre il minor quantitativo di software all'attaccante perché più se ne espone più c'è probabilità che contenga dei bachi che possono essere sfruttati per eseguire un attacco:

- Se si fa sicurezza a livello SSL l'attaccante potrà andare a sfruttare solamente i bachi contenuti all'interno di tale libreria; di conseguenza si avrà una superficie d'attacco molto piccola.
- Se invece si sceglie di fare SSL con server authentication non solo l'attaccante potrà sfruttare i bachi contenuti all'interno della libreria SSL, ma potrà andare a sfruttare anche i bachi contenuti all'interno del server Web. Di conseguenza la superficie di attacco sarà cresciuta.
- Fare sicurezza all'ultimo livello permette all'attaccante di sfruttare tutti i bachi contenuti nella libreria SSL, nel server Web e nel codice delle pagine implementate dallo sviluppatore applicativo.

Quello che si deve cercare di fare quindi è quello di fare l'autenticazione nel livello più basso possibile. Purtroppo questo molto spesso non è possibile perché gli utenti di rado hanno un certificato X.509, o anche nel caso in cui ce lo avessero non è detto che il sistemista abbia attivato la funzione di mapping delle credenzialità. Per quanto riguarda l'autenticazione a livello HTTP, molto spesso la basic o la digest authentication non sono attivate dagli sviluppatori applicativi (soprattutto per ignoranza). Quindi in generale l'autenticazione viene eseguita utilizzando dei form inseriti direttamente all'interno delle pagine Web, perché questa è la soluzione più comoda per gli sviluppatori applicativi.

Username e password in un form?

Tecnicamente non è importante la sicurezza contenuta all'interno della pagina in cui si introduce i dati, perché la sicurezza effettiva dipende dalla URI del metodo usato per inviare username e password. Questo significa che la pagina può anche essere trasmessa usando HTTP (http://www.ecomm.it/login.html), ma le credenziali dell'utente devono necessariamente essere inviate utilizzando HTTPS (`<form ... action="https://www.ecomm.it/login.asp">`).

Tuttavia procedere in questo modo è un errore. Da un punto di vista psicologico è importante la sicurezza della pagina in cui si introducono i dati perché pochi utenti hanno le conoscenze tecniche necessarie a verificare la URI del metodo usato per l'invio.

7.5 Sistemi di pagamento elettronico

I sistemi di pagamento elettronico sono sempre più importanti perché ormai tantissime transazioni di acquisto vengono fatte tramite canali Web.

In passato c'è stato un tentantivo di un professore olandese di sviluppare la moneta elettronica perfetta (da un punto di vista scientifico), chiamata DIgiCash, ma questo sistema è stato un fallimento a causa di problemi tecnici e normativi. I governi hanno da sempre il terrore della moneta elettronica perché così come il denaro anche la moneta elettronica non è tracciabile, e soprattutto si corre il rischio che qualcuno crei della moneta virtuale senza avere la corrispondente moneta reale, con un alto rischio di inflazione.

Oggigiorno tutti i governi preferiscono che i pagamenti avvengano in modo elettronico tramite l'utilizzo delle carte di credito, perché questo permette di avere un controllo perfetto del cittadino sapendo che cosa ha acquistato e in quale momento.

Il metodo di pagamento con carta di credito avviene tramite la trasmissione del numero della carta su di un canale SSL; l'utilizzo di un canale sicuro garantisce la protezione della trasmissione

del numero della carta durante il transito in rete, ma non fornisce nessuna garanzia contro le frodi. Qualche anno fa VISA Europa ha dichiarato che il 50% dei tentativi di frode nascono da transazioni Internet, che però erano solo il 2% del suo volume d'affari.

Una frode può ad esempio avvenire quando si è al ristorante, e al momento di saldare il conto si fornisce la propria carta di credito al cameriere. Questo, se non è controllato, potrebbe copiare i numeri presenti sulla carta, e potrebbe successivamente usarli per fare acquisti su Internet. Internet quindi diventa uno dei mezzi principali sui quali avvengono delle frodi. In passato se si clonava una carta occorreva andare in un negozio per fare acquisti, mentre ora è possibile acquistare qualunque cosa tranquillamente stando a casa.

7.5.1 Sicurezza delle transazioni basate su carta di credito

Negli anni ci sono stati più tentativi da parte dei maggiori fornitori di carte d credito di creare delle soluzioni più sicure. In particolare:

- *Secure Transaction Technology (STT)*, nato da una collaborazione di VISA con Microsoft.
- *Secure Electronic Payment Protocol (SEPP)*, creato da Mastercard, IBM, Netscape e altri.
- *Secure Electronic Transaction (SET)*, nato dalla fusione di STT con SEPP.

Secure Electronic Transaction (SET)

SET non è un sistema di pagamento, ma è un insieme di protocolli per usare in una rete aperta in modo sicuro l'infrastruttura già esistente dei pagamenti con carta di credito.

Esso è caratterizzato da due principali caratteristiche:

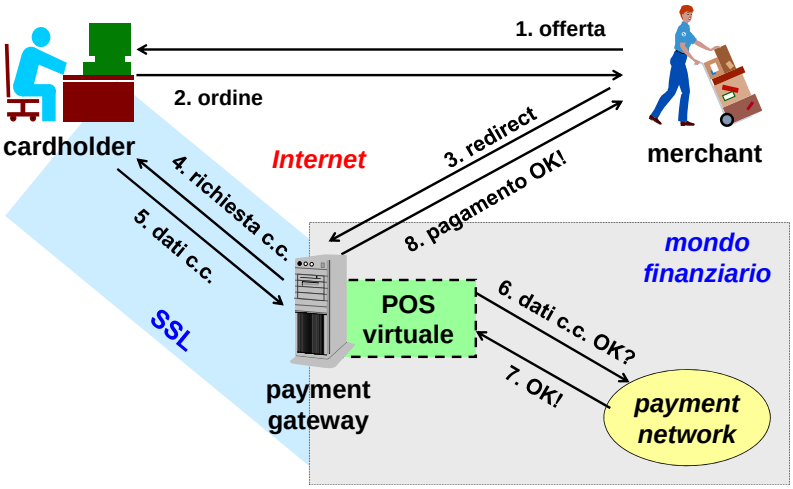
- Usa certificati X.509v3 dedicati esclusivamente alle transazioni SET, in modo da garantire la sicurezza dei vari elementi. Tali certificati erano molto cari, e ha rappresentato fin da subito un ostacolo per chi volesse iniziare ad effettuare delle vendite via Internet.
- SET è stato pensato in modo tale da assicurare la privacy degli utenti perché mostra a ciascuna parte solamente i dati che le competono.

Tecnicamente SET è uno standard molto valido, ma presenta due gravi errori di progettazione:

- Mettere un prezzo di ingresso molto alto per l'acquisto di un certificato X.509v3.
- Basarsi sul fatto che l'utente dovesse avere sul suo pc un cosiddetto *SET wallet*, ossia un software che conteneva chiavi pubbliche, chiavi private, i pagamenti effettuati e così via. Questo è da considerarsi un fallimento perché i pagamenti elettronici devono assolutamente basarsi sull'utilizzo di un browser, e non obbligare gli utenti ad effettuare acquisti tramite un software; quest'ultimo inoltre ha anche un problema di gestione, in quanto tutti i problemi che gli utenti avranno dovranno essere risolti dagli sviluppatori di tale software.

Tutte queste caratteristiche hanno portato al fallimento di questo standard, che ora è pressoché in disuso.

Architettura di pagamento via Web



Innanzitutto si può notare il mondo finanziario (in grigio), separato dal resto di Internet. Il problema sta proprio nel capire sul come interfacciare i metodi di pagamento che già esistono con Internet.

1. Il *merchant* offre una serie di prodotti tramite il suo canale di vendità online, pagabili tramite carta di credito. Dopo aver consultato il catalogo dei prodotti, il *cardholder* decide di effettuare un acquisto. A questo punto il merchant invia formalmente al cardholder un'*offerta*.

2. Il cardholder risponde accettando l'offerta. A questo punto alcuni commercianti richiedono all'utente di inserire all'interno di un form i dati della propria carta di credito, e dopo aver eseguito tutti i controlli necessari si potrà procedere al pagamento vero e proprio. Questa è una cosa perisolosissima, perché significa che i dati della carta di credito vengono trattati direttamente dal merchant, il quale se non è una persona fidata potrebbe usare i dati forniti dall'utente per effettuare una frode.

3. In generale la soluzione più usata per i pagamenti online è quella di usare un *payment gateway*. Il merchant non gestisce direttamente i dati della carta di credito, ma nel momento in cui gli arriva una richiesta di acquisto viene effettuato un *redirect* verso un circuito finanziario con il quale ha stretto un accordo. Il pagamento quindi non viene fatto al merchant, ma viene fatto al circuito finanziario.

4. Una volta ricevuta una richiesta di pagamento, la prima operazione eseguita dal payment gateway è quella di aprire un canale SSL con il cardholder. Una volta creato il canale sicuro si chiede all'utente di inserire i dati della carta di credito.

5. Il cardholder come da richiesta provvede ad inserire i dati della sua carta di credito.

6. A questo punto il payment gateway svolge la sua funzione di gateway, ossia garantisce un collegamento tra il mondo di Internet e il mondo finanziario. In particolare il gateway attiva un software detto *POS virtuale*, usato per trasformare i dati ricevuti in una transazione adatta al mondo finanziario.

7. La normale rete di pagamento, dopo aver effettuato tutti i controlli necessari, risponde con un messaggio positivo.

8. A questo punto il gateway risponderà al merchant dicendogli che il pagamento è andato a buon fine, e quindi l'ordine può considersi concluso.

Rispetto a SET con questo schema non si ha una protezione della privacy in quanto il payment gateway conoscerà sia la merce che si sta cercando di acquistare, sia i dati della carta di credito. Questo chema però fornisce una protezione verso il merchant, che non ha più bisogno di sapere i dati della carta di credito del cardholder ma ha solo informazioni per quanto riguarda la merce acquistata.

Questo schema si basa sulle ipotesi che l'acquirente possiede una carta di credito, e che esso stia utilizzando un browser con SSL. Anche se si utilizza un canale sicuro, il livello di sicurezza effettivo dipende dalla configurazione sia del server sia del client.

Payment Card Industry Data Security Standard (PCI DSS)

PCI DSS è lo standard per la sicurezza dei dati della Payment Card Industry, ossia di tutti gli enti che usano carte di pagamento.

Oggi è uno standard obbligatorio richiesto da tutti quanti gli enti finanziari per permettere l'uso delle loro carte ai fini di transazioni via Internet.

Questo standard è molto più prescrittivo (ossia da delle regole molto precise) rispetto ad altre norme di sicurezza.

Per poter essere definiti PCI DSS compliant occorre rispettare tutta una serie di requisiti:

- *Costruire e mantenere una rete protetta*
  1. Installare e mantenere una configurazione con firewall per proteggere i dati dei titolari delle carte.
  2. Non usare password di sistema predefinite o altri parametri di sicurezza impostati dai fornitori.
- *Proteggere i dati dei titolari delle carte*
  3. Proteggere i dati dei titolari delle carte memorizzati.
  4. Cifrare i dati dei titolari delle carte quando trasmessi attraverso reti pubbliche aperte.
- *Rispettare un programma per la gestione delle vulnerabilità*
  5. Usare e aggiornare con regolarità l'antivirus.
  6. Sviluppare e mantenere applicazioni e sistemi protetti.
- *Implementare misure forti per il controllo accessi*
  7. Limitare l'accesso ai dati dei titolari delle carte solo se effettivamente indispensabili per lo svolgimento dell'attività commerciale.
  8. Dare un ID univoco ad ogni utente del sistema informativo.
  9. Limitare la possibilità di accesso fisico ai dati dei titolari delle carte
- *Monitorare e testare le reti con regolarità*
  10. Monitorare e tenere traccia di tutti gli accessi effettuati alle risorse della rete e ai dati dei titolari delle carte.
  11. Eseguire test periodici dei processi e dei sistemi di protezione.
- *Adottare una Politica di Sicurezza*
  12. Adottare una Politica di Sicurezza.

Capitolo 8

XACML e SAML

Le applicazioni web moderne oggi utilizzano due standard per effettuare il controllo accessi, e quindi l'autenticazione e l'autorizzazione: XACML e SAML.

8.1 XACML

Il primo standard, definito da OASIS (organizzazione indipendente che sviluppa gli standard per XML nel mondo web), è chiamato *eXtensible Access Control Markup Language (XACML)*. Questo è un linguaggio, basato su XML, che serve a descrivere le politiche di autorizzazione attraverso la definizione di:

- Subject, che possono essere gli utenti del sistema, ma anche i computer che agiscono nel sistema e i servizi offerti.
- Resource (documenti, i file, i dati), tutti identificati tramite delle URI.

XACML è un linguaggio che serve per gestire accessi a risorse che sono protette da specifiche politiche di autorizzazione. A questo scopo lo standard definisce il formato dei dati per fare una richiesta di accesso e per fornire una risposta a tale richiesta, ma non viene peò fornita nessuna indicazione su quale protocollo occorre usare per il trasferimento di tali richieste/risposte. Si è quindi liberi di scegliere il protocollo client-server da utilizzarsi.

8.1.1 Controllo accessi policy-based

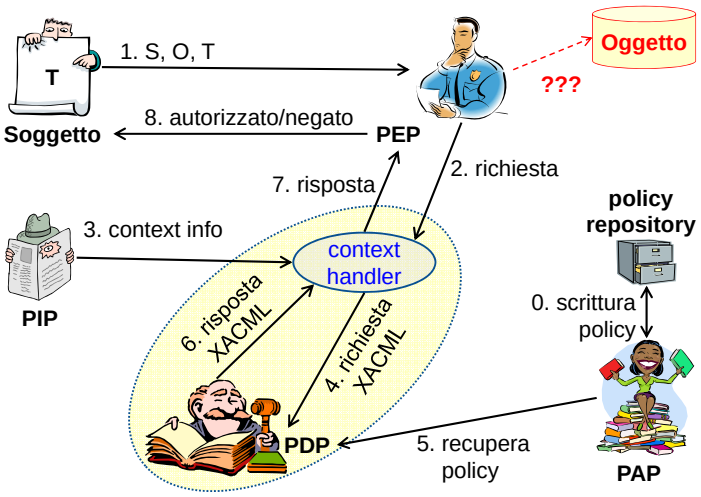
In generale XACML è l'ultima incarnazione di una filosofia più generale, il cosiddetto *controllo accesso Policy-Based*, ossia le regole non sono definite utente per utente ma vengono definite delle politiche di sicurezza generali.
I meccanismi Policy-Based sono stati definiti dall'IETF molto tempo fa per decidere quali erano le politiche da usarsi per fare QoS sui router. I concetti sono stati poi generalizzati ed estesi alla gestione dei sistemi informativi (dall'organismo chiamato DMTF) e soprattutto al controllo degli accssi in ambienti distribuiti (OASIS). Quest'ultima parte oggi è diventata fondamentale perché il web2.0 ha come punto cardine l'integrazione di più servizi diversi, offerti da enti diversi, e quindi con politiche di sicurezza differenti; avere quindi un sistema che definisca a grandi linee quali sono le politiche di sicurezza da usare, da implementarsi da tutti quanti gli attori del sistema, è estremamente importante.

Componenti controllo accessi policy-based

- *Policy Enforcement Point (PEP)*: elemento che protegge una risorsa e ne permette l'accesso solo se le verifica di compatibilità con la policy è positiva.

- *Policy Decision Point (PDP)*: elemento a cui il PEP si rivolge per sapere se deve dare o meno accesso ad una certa risorsa. In particolare esso riceve tutte le informazioni necessarie (policy, soggetto richiedente, risorsa richiesta, tipo di accesso, contesto) per decide se concedere o meno l'accesso.
- *Policy Information Point (PIP)*: fornisce tutte le informazioni relative all'accesso richiesto.
- *Policy Access Point (PAP)*: fornisce la policy applicabile all'accesso richiesto.

8.1.2 XACML - Schema di accesso



Nell'esempio si ha un soggetto S, che vuole accedere ad un oggetto O facendo una transazione di tipo T. L'oggetto è però protetto da un PEP, e prima di fornire accesso all'utente sarà necessario effettuare un controllo per vedere se quest'ultimo ne ha diritto.

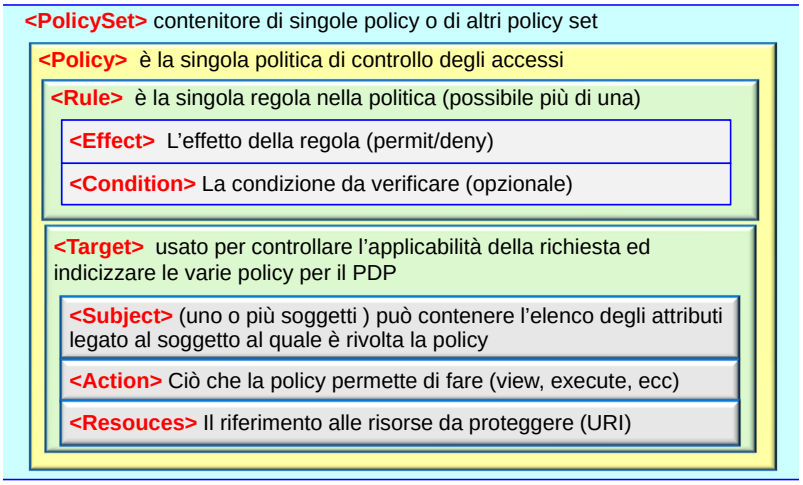
Il PEP provvede a questo punto a contattare il PDP. PEP e PDP potrebbero dover utilizzare un context handler per poter comunicare correttamente (questo perché solitamente il PEP è un firewall, e questo in generale non parla XACML).
Il fatto che la richiesta da parte del PEP al PDP possa richiedere una traduzione potrebbe essere utile perché facendo uso del PIP è possibile aggiungere alla richiesta informazioni di contesto (se ad es. la richiesta avviene da un device mobile si potrebbe aggiungere la posizione in cui tale richiesta è effettuata). Tali informazioni molto spesso sono indicate bella forma di asserzioni SAML.

Una volta ricevuta la richiesta il PDP deve prendere una decisione, ma per farlo necessità di sapere qual è la politica di sicurezza applicabile all'oggetto O. A tale scopo il PEP utilizza il PAP per recuperare la policy, la quale sarà contenuta all'interno di un policy repository (al passo zero si è avuta la definizione di tutte le policy, con l'indicazione per ciascun oggetto di quale policy da seguire).

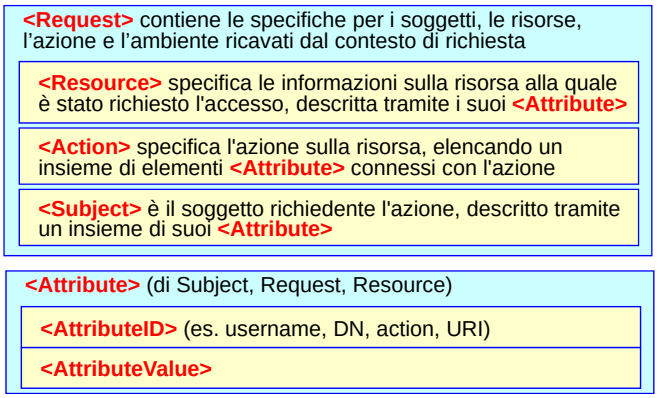
A questo punto il PDP sapendo la richiesta che è stata fatta, avendo le informazioni di contesto e sapendo la policy che si applica può prendere una decisione, e fornisce una risposta in formato XACML che solitamente dovrà essere tradotta prima di poter essere inviata al PEP.

Ricevuta la risposta il PEP potrà dire al soggetto se l'accesso è stato autorizzato o se invece è stato negato.

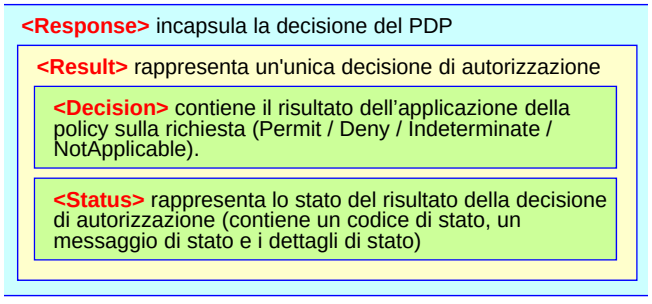
8.1.3 XACML - Formato policy



8.1.4 XACML - Formato richiesta



8.1.5 XACML - Formato risposta



8.2 SAML

Security Assertion Markup Langage (SAML) è un formato dati che serve a:

- Esprimere vari tipi di affermazioni.
- Formulare richieste di affermazioni.

- Esprimere risposte contenenti tali affermazioni.

Un'affermazione si basa su un oggetto SAML che prende il nome *assertion*.

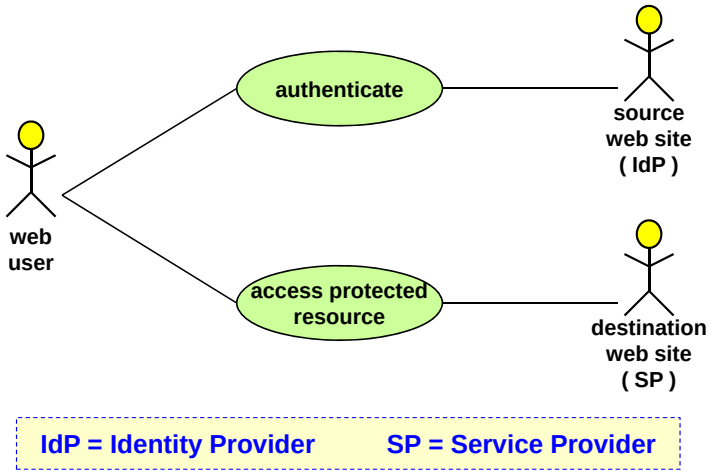
Lo scopo di questo linguaggio è quello di rendere standard e semplificare tutte le interazioni che sono relative a stabilire dei permessi (generi, non necessariamente solo di accesso) in un sistema distribuito multi-dominio.

Come XACML anche SAML è uno standard OASIS basato su sintassi XML.

8.2.1 SAML - Versioni

- SAML 1.0
  - Versione base del nov' 2002.
- SAML 1.1
  - Versione del set' 2003, ha introdotto la protezione dei messaggi tramite l'uso della firma digitale XML (XML-dsig).
  - Inoltre ha anche definito i profili per il web browser SSO. In particolare ha definito due profili: browser/artifact profile, nel quale i dati non sono trasmessi direttamente ma viene solo indicato dove si trovano (token SAML by ref), e browser/POST profile, nel quale i dati sono trasmessi dentro il token SML (token SML by value).
    Questi due profili hanno molta importanza a seconda dello schema implementativo che si vuole utilizzare. Se ad esempio si suppone che SAML venga usato da una persona dotata di un PC e di un collegamento veloce, allora va benissimo usare il metodo POST che è decisamente più semplice da implementare; se invece si suppone di avere un utente dotato di un dispositivo non molto potente e/o un sistema di rete non particolarmente veloce allora conviene usare il profilo artifact in cui si invia solo un puntatore ai dati.
- SAML 2.0
  - Versione più recente dello standard del mar' 2005.
  - Ancora oggi non è totalmente supportato principalmente per il fatto che è incompatibile con le versioni precedenti.
  - Anche questa versione permette di proteggere i messaggi tramite XML-dsig.
  - Inoltre aggiunge la possibilià di cifrare i dati sfruttando XML-enc. In particolare è possibile cifrare gli identificatori, gli attributi e le asserzioni (i valori dati agli attributi).
  - Vengono anche definiti nuovi protocolli di trasporto, di binding e di profili.

8.2.2 Web browser SSO use case

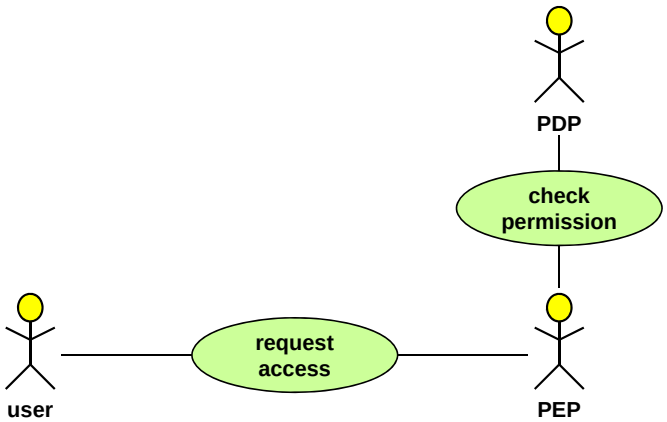


Tipicamente quando si parla di SAML, gli attori che entrano in gioco oltre al PEP e al PDP sono l'idP (Identity Provider) e l'SP (Service Provider).

In questo use case si ha un utente che desidera usurfruire di un servizio fornito da un certo SP. Per usare tale servizio però è necessario autenticarsi. L'idea è quella di non autenticarsi direttamente all'interno del sito web contenente tale servizio, ma quella di usare un'autenticazione fornita da un sito terzo (IsP). Si pensi ad esempio a quando si effettua il login presso un sito sfruttando le credenziali di Facebook o di Google.

In altre parole lo schema prevede che l'utente si autentichi prima all'interno dell'IdP, il quale rilascerà un'asserzione di autenticazione all'SP affermando che il possessore di tale token si è appena autenticato con successo. In questo caso quindi SAML non viene usato sono per autenticarsi ma anche per veicolare l'affermazione che l'autenticazione ha avuto successo.

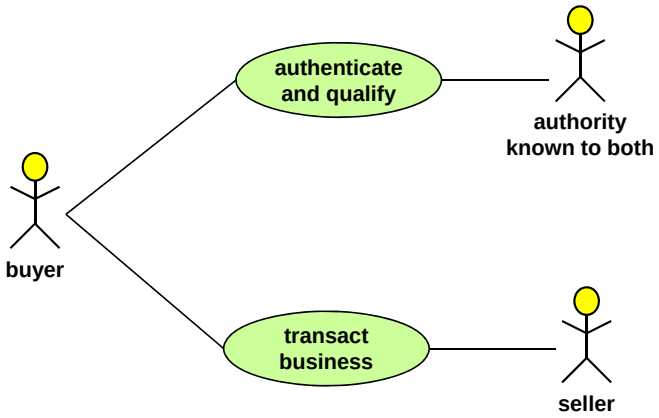
8.2.3 Authorization service use case



Nell'authorization service use case si ha un utente che desidera accedere ad una certa risorsa, la quale è però protetta dal PEP. Prima di poter autorizzare o negare l'accesso il PEP dovrà interogare il PDP per controllare se l'utente possiede o meno i diritti di accesso a tale risorsa.

La risposta del PDP viene veicolata sotto forma di un'asserzione SAML con firma digitale, con la quale si autorizza (o si nega) l'accesso ad una certa risorsa da parte dell'utente. Si tratta di un'asserzione di autorizzazione.

8.2.4 Back office transaction use case



Nel back office transaction si ha un utente che deve fare un acquisto presso un venditore in nome e per conto della propria azienda. A tale scopo l'utente si rivolge ad un'autorià nota sia a lui che al venditore.

Tale autorità ha lo scopo di emettere un'asserzione con la quale non solo il compratore viene autenticato, ma esso viene anche autorizzato a fare acquisti in nome e per conto della sua azienda. In particolare l'asserzione viene usata non tanto per fare autenticazione, il compratore potrebbe anche essere anonimo; lo scopo dell'asserzione è quello di autorizzare un certo soggetto possessore del token ad eseguire una determinata operazione.

8.2.5 SAML assertion

L'asserzione è una dichiarazione relativa ad un fatto pertinente ad un soggetto, ad es. il ruolo ricoperto da un utente. Tale dichiarazione viene fatta da un certo issuer.
I tre tipi base di asserzione sono:

- Autenticazione.
- Attributi posseduti.
- Decisione di autorizzazione.

Lo schema SAML essendo basato su XML è estensibile, e quindi è possibile aggiungere altri tipi di assertion.

Visto che l'asserzione è un oggetto molto importante può essere firmato digitalmente tramite lo schema XML signature, proprio per garantire integrita dei dati in esso contenuti e per sapere l'identità dell'issuer.

Info comuni a tutte le assertion

Tutte quante le asserzioni di qualunque tipo contengono alcuni elementi comuni:

- L'identificativo dell'issuer e la data e ora in cui è stata creata l'asserzione.
- Un identificatifo dell'asserzione, assertion ID.
- Il soggetto. Visto che si sta parlando di soggetti a multi-dominio, il soggetto tipicamente sarà un nome più il dominio di sicurezza in cui quel nome è valido.
- Eventuali condizioni sotto cui l'asserzione è valida
  - I client SAML devono rifiutare asserzioni che contengono delle condizioni di validità che non siano comprese.

  - Una condizione molto importante solitamente presente è l'assertion validity period, ossia la durata dell'asserzione.

- Altre informazioni utili, come ad esempio una spiegazione/prova della base su cui è stata costruita l'asserzione.

Authentication assertion

L'authentication assertion è quella che un issuer emette per affermare che un certo soggetto S è stato autenticato con il meccanismo M in un determinato istante di tempo T.

Si noti che SAML non effettua direttamente l'operazione di autenticazione, ma essa è fatta usando ad esempio uno schema username/password, sfida e risposta, chavi asimmetriche e così via. SAML serve solamente a fornire un meccanismo per creare un collegamento tra il risultato di autenticazione e il soggetto che dovrà sapere tale informazione.

```
<saml:Assertion
   MajorVersion="1" MinorVersion="0"
   AssertionID="192.168.1.1.12345678"
   Issuer="Politecnico di Torino"
   IssueInstant="2007-12-03T10:02:00Z">
   <saml:Conditions
      NotBefore="2007-12-03T10:00:00Z"
      NotAfter="2007-12-03T10:05:00Z" />
   <saml:AuthenticationStatement
      AuthenticationMethod="password"
      AuthenticationInstant="2007-12-03T10:02:00Z">
      <saml:Subject>
         <saml:NameIdentifier
            SecurityDomain="polito.it" Name="alioy" />
      </saml:Subject>
   </saml:AuthenticationStatement>
</saml:Assertion>
```

Nella seguente asserzione di esempio si afferma di avere autenticato con il metodo *password*, nell'istante 2007-12-03 alle ore 10.02, il soggetto identificato con il nome alioy all'interno del dominio di sicurezza polito.it.

Attribute assertion

L'attribute assertion è quella in cui un certo issuer afferma che il soggetto S è associato a degli attributi A, B, C, ... che hanno attualmente i valori "a", "b", "c", ...

Tipicamente questa asserzione è ottenuta tramite una query LDAP o altri meccanismi di lookup in un DB.

Ad esempio un'affermazione potrebbe essere: il soggetto identificato con il nome "alioy" del dominio "polito.it" è associato all'attributo "Dipartimento" con valore "DAUIN". Come si può notare dall'esempio che segue un certo attributo ha validità solamente all'interno di un dominio ben definito (AttributeNameSpace).

```
<saml:Assertion ...>
   <saml:Conditions .../>
   <saml:AttributeStatement>
      <saml:Subject>
         <saml:NameIdentifier
            SecurityDomain="polito.it"
            Name="alioy" />
      </saml:Subject>
      <saml:Attribute
         AttributeName="Dipartimento"
         AttributeNamespace="http://polito.it">
         <saml:AttributeValue>
            DAUIN
         </saml:AttributeValue>
      </saml:Attribute>
   </saml:AttributeStatement>
</saml:Assertion>
```

Authorization decision assertion

Quest'ultima asserzione viene utilizzata da un issuer per affermare di aver preso una decisione circa una richiesta da parte di un soggetto S per un accesso di tipo T alla risorsa R, basandosi su una certa evidenza E.

Il soggetto S può essere un individuo, ma può anche essere un programma.

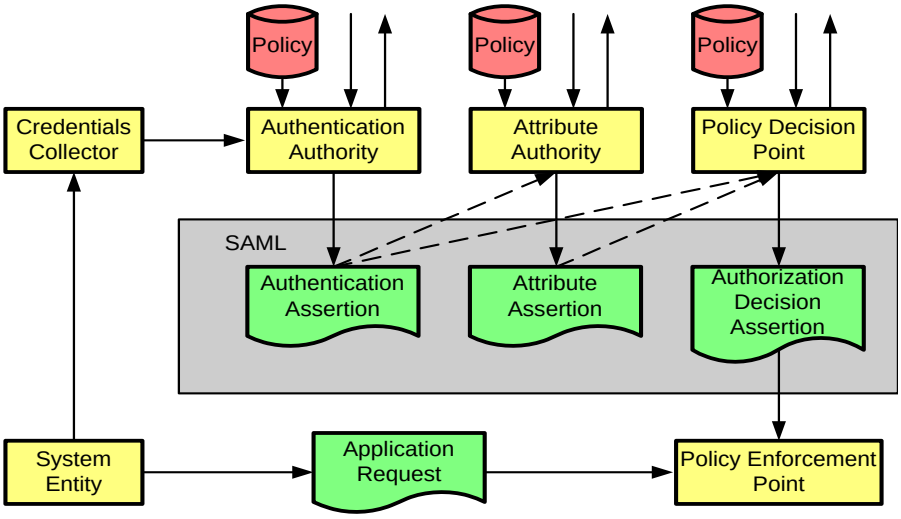
La risorsa potrebbe ad esempio essere una pagina web, un file, l'invocazione di un webservice, ...

```
<saml:Assertion ...>
   <saml:Conditions .../>
   <saml:AuthorizationStatement
      Decision="Permit"
      Resource="http://did.polito.it/m2170.php">
      <saml:Subject>
         <saml:NameIdentifier
            SecurityDomain="polito.it" Name="alioy" />
      </saml:Subject>
  </saml:AuthorizationStatement>
</saml:Assertion>
```

8.2.6 SAML - Modello produttore-consumatore

In generale SAML implementa un modello produttore-consumatore, nel senso che c'è qualcuno che produce le asserzioni e c'è qualcuno che le consuma per effettuare una determinata operazione.

In figura si hanno i tre tipi di asserzioni possibili, con l'indicazione di chi potrebbe essere il loro possibile produttore e consumatore.

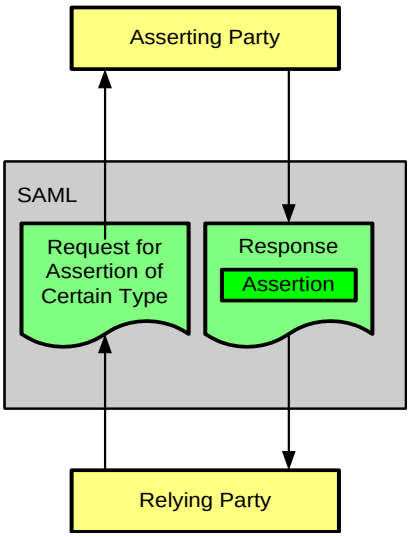


Innanzitutto un'authentication assertion viene prodotta da una *Authentication Authority (AA)*, e viene consumata da una *Attribute Authority*; questo perché prima di assegnare un certo attributo ad un soggetto occorre sapere l'identità di tale soggetto. In base a questa asserzione l'Attribute Authority emette un attribute assertion, che può essere utilizzata da un *Policy Decision Point* per prendere una decisione circa l'autorizzazione di un soggetto a svolgere una certa operazione. In questo caso quest'ultima asserzione verrà consumata da un *Policy Enforcement Point (PEP)*, ossia l'entità che decide se un soggetto è autorizzato o meno ad avere accesso ad una certa risorsa.

Ognuno di questi sistemi si basa su delle policy precise: come autenticare gli utenti, quali utenti hanno quali attributi e quali sono le politiche di accesso.

In generale il processo di emissione di un'asserzione inizia in seguito ad una richiesta applicativa di un *System Entity*, la quale è stata temporaneamente bloccata dal PEP. A questo punto il System Entity dovrà interagire con un'entità che raccoglie le credenziali di autenticazione (*Credetials Collector*), con i quali sarà possibile iniziare il processo di creazione dell'asserzione.

8.2.7 SAML - Protocollo per le assertion



Per veicolare le asserzioni tipicamente esse vengono messe dentro un elemento protocollare di tipo SAML. Ci possono essere gli elementi protocollari di risposta, che quindi contengono le

asserzioni vere e proprie, e quelli che invece contengono la richiesta di ricevere un certo tipo di asserzione.

Il *Relying Party* è l'entità che fa la richiesta verso un *Asserting Party*. In particolare il Relying Party chiede informazioni circa un soggetto, e l'Asserting Party provvederà a produrre la risposta che verrà consumata dallo stesso Relying Party.

Richiesta di authentication assertion

Una richiesta di authentication assertion è concettualmente qualcosa del tipo "Per favore, forniscimi l'informazione di autenticazione relativa a questo soggetto, se ne hai".

Si noti che si sta assumendo che il richiedente e il risponditore abbiano una qualche relazione di fiducia:

- Devono avere in comune il fatto di parlare dello stesso Soggetto e
- La risposta è da considerarsi come una "lettera di introduzione" per il Soggetto.

```
<samlp:Request
  MajorVersion="1" MinorVersion="0"
  RequestID="128.14.234.20.12345678" >
  <samlp:AuthenticationQuery>
    <saml:Subject>
      <saml:NameIdentifier
        SecurityDomain="polito.it" Name="alioy" />
    </saml:Subject>
  </samlp:AuthenticationQuery>
</samlp:Request>
```

L'asserzione inizia con la stringa *samlp*, che sta ad indicare il protocollo usato per veicolare asserzioni SAML. In particolare nell'esempio si ha una samlp di tipo request, in cui si sta facendo una query di autenticazione: il richiedente vuole sapere se l'utente con il nome "alioy" è registrato all'interno del dominio "polito.it".

Relazione di fiducia

La risposta deve anche veicolare una relazione di fiducia, nel senso che chi accetta l'asserzione deve in qualche modo fidarsi di chi genera l'asserzione.

Nella pratica la relazione di fiducia può essere stabilita in due modi diversi:

- Si può avere una fiducia diretta, basata ad esempio su un meccanismo di push o pull su un canale sicuro (es. SSL). Questo significa che colui che autentica e colui che userà tale autenticazione usano un canale sicuro per il trasferimento della richiesta e della risposta.
- Nel caso in cui la risposta non viene trasferita direttamente dall'emettitore al richiedente, ma passa tramite l'utente, allora è meglio che la risposta sarà firmata con una chiave condivisa o pubblica.

Binding SAML

Binding SAML è quell'insieme di regole che definiscono che cosa trasportare (what) e come trasportarlo (how). In altre parole è un protocollo di rete per richieste e risposte SAML.

In SAML 1.0 come protocollo di binding era definito SAML/SOAP-over-HTTP. Nelle versioni successive, e soprattuto in SAML 2.0 sono stati introdotti altri tipi di binding:

- SAML SOAP binding, basato su SAML 1.1, in modo da avere compatibilità all'indietro.

- Reverse SOAP (PAOS) binding.
- HTTP redirect (GET) binding.
- HTTP POST binding.
- HTTP artifact binding.
- SAML URI binding.

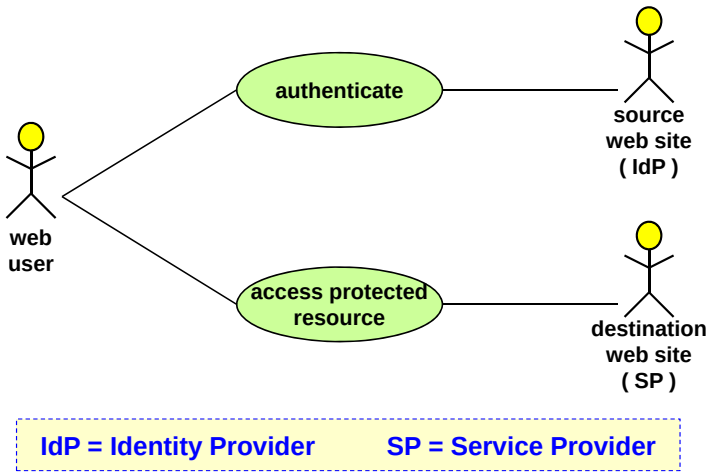
I binding oggi più usati sono tutti quelli basati direttamente su HTTP.

Profili SAML

Un profilo SAML definisce esattamente come si deve usare SAML per un particolare use case. Ad esempio:

- "Web browser profile" definisce come usare SAML per implementare SSO web.
- "SOAP profile" serve per fare asserzioni relative ad un payload SOAP.

SSO push use case



1. [GET service URI] Il browser va dall'SP ed esegue una GET alla URI presso cui vuole avere un servizio.
2. [REDIRECT to IdP with SAML-authN-req] L'SP non accetta la richiesta di accesso perché non conosce l'identità del richiedente. L'SP quindi risponde al browser inviandogli un comando di REDIRECT verso un certo IdP in modo da autenticarsi, a cui sarà allegata la SAML authentication request (SAML-authN-req).
3. [GET with SAML-authN-req] A questo il browser inoltre la richiesta di autenticazione all'IdP tramite una GET.
4. [HTML form: POST to SP with hidden fieldcontaining SAML-authN-resp] Come risposta l'IdP invia al browser un form HTML in cui sarà necessario inserire username e password. Se i dati inseriti saranno corretti, allora l'IdP fornirà una risposta contenente un campo nascono con la SAML authentication response.
5. [POST with hidden SAML-authN-resp] Ricevuta la risposta il browser ne effettua il post verso il SP.

6. [verifies SAML-authN-resp and eventually provides requested service] L'SP verifica la validità della risposta (eseguendo un controllo della firma digitale o calcolando un MAC), e nel caso in cui fosse validità fornirà il servizio richiesto.

Questo use case viene anche definito *front-channel exchange*, nel senso che tutte le informazioni passato direttamente tramite le interfacce frontali.

SSO pull use case

I passaggi 1, 2 e 3 sono uguali al precedente use case.

4. [HTML form: POST to SP with artifact (=pointer to SAML-authN-resp stored at IdP)] Una volta che l'IdP ha ricevuto la richiesta di autenticazione da parte del browser, esso crea un form facendo un POST verso l'SP usando un artifact. Quest'ultimo altro non è che un puntatore alla risposta della richiesta generata dall'SP, e successivamente inoltrata dal browser.

5. [POST with artifact] Ricevuta la risposta dall'IdP, il browser provvederà ad inoltrarla all'SP.

6. [GET with artifact] L'SP ricevuto l'artifact si mette in collegamento direttamente con l'IdP, in modo da poter avere la risposta vera e propria.

7. [SAML-authN-resp] L'IdP trasmetterà all'SP la risposta associata a quell'artifact.

7. [verifies SAML-authN-resp and eventually provides requested service] L'SP verifica la risposta e in caso positivo fornisce il servizio.

Visto che in questo caso la risposta è presa direttamente dall'IdP, potrebbe non essere necessario che essa sia firmata (a condizione che la comunicazione tra SP e IdP avvenga tramite un canale sicuro). Questo perché la risposta non è passata tramite una terza parte che potrebbe averla manipolata.

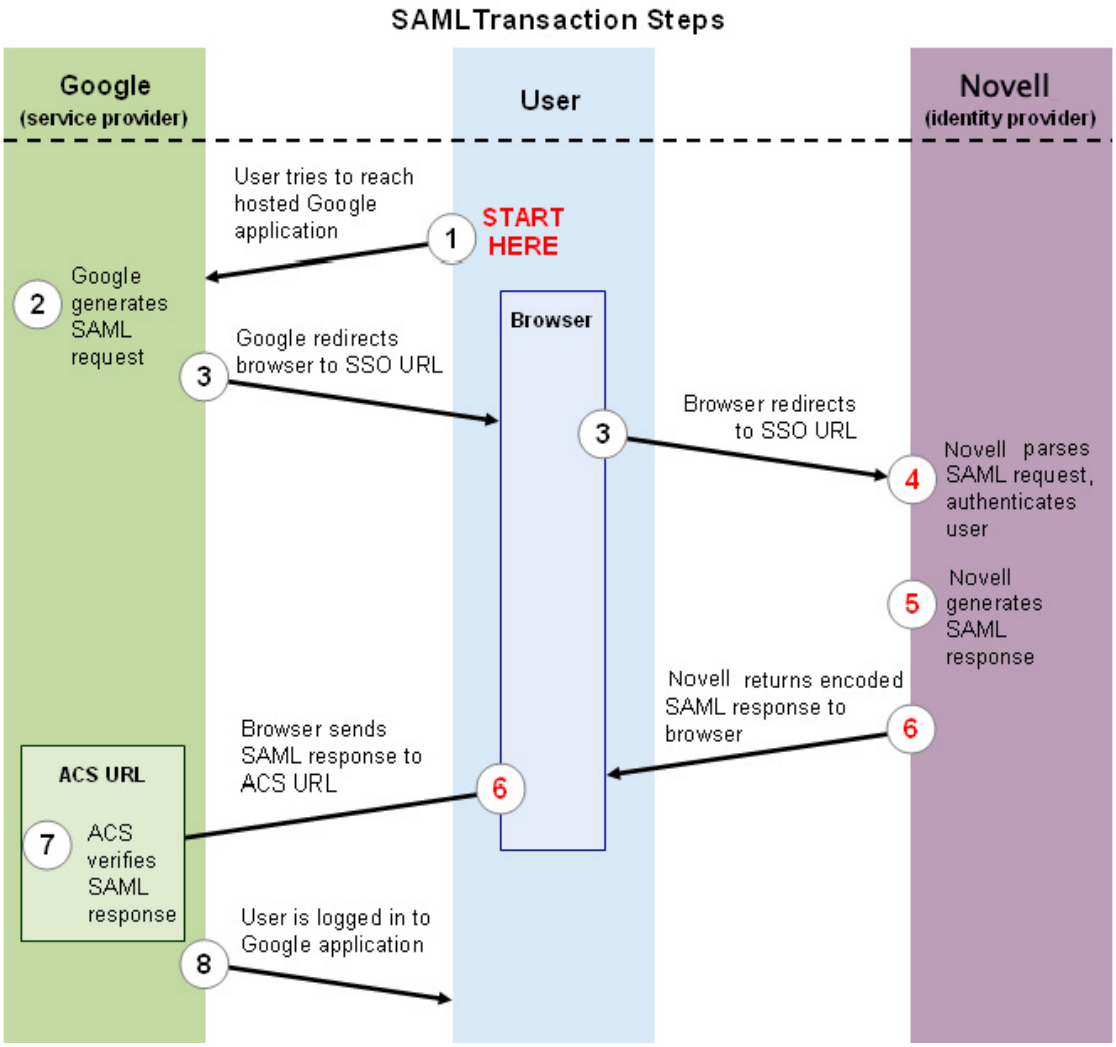
Questo schema, chiamato *back-channel exchange*, è quello da preferirsi in caso di device mobile, e nei casi dove la risposta è molto grande.

8.2.8 SAML SSO per Google Apps

Il fatto che SAML oggi sia largamente usato è dovuto principalmente al fatto che è stato usato da Google per le sue Apps per effettuare il SSO.

Si immagini una ditta (parter) che decide di non avere più un'applicazione sul proprio server, ma che decide di implementarla sui server di Google come una Google App; in questo scenario Google diventa un service provider.
In ogni caso l'azienda parter vuole mantenere il controllo della parte di autenticazione ed autorizzazione, e quindi di svolgere la funzione di Identity Provider (IdP).

Lo scambio è basato su SAML 2.0 con una firma XML.



1. L'utente cerca di accedere all'applicazione ospitata sui server Google.
2. L'accesso a tale applicazione però è consentito solamente ai dipendenti dell'azienda. A tale scopo Google genera una SAML request, con la quale si vuole verificare l'identità dell'utente.
3. Google invia la richiesta di autenticazione al browser dell'utente, con il quale lo re-indirizza verso un indirizzo che fornisce il servizio di SSO; tale indirizzo è ospitato all'interno del server aziendale.
4. Ricevuta la richiesta, l'azienda partner ne esegue il parsing e provvede ad eseguire l'autenticazione dell'utente.
5. L'azienda parter genera la SAML response, che indica non soltanto che l'utente è autenticato ma che ha anche il diritto di accedere al servizio richiesto.
6. La risposta è codificata ed è inviata al browser dell'utente, il quale ne eseguirà il redirect verso la cosiddetta Assertion Consumer Service (ACS), ossia il servizio che deve consumare le asserzioni.
7. L'ACS effettua una verifica la SAML response basandosi sulla firma digitale (firma generata usando un certificato che l'azienda parter ha fornito a Google al momento della sottoscrizione del servizio).

8. Se la risposta ha dato esito positivo, l'utente avrà diritto ad usare il servizio richiesto.

Caratteristiche del servizio

Il partener deve fornire a Google la URL su cui fare SSO e il certificato X.509 per verificare le proprie firme.

Il passo 3 contiene in modo opaco (e quindi non leggibile chiaramente dal browser):

- La URL del servizio Google richiesto dall'utente.
- La richiesta SAML di autenticazione.
- La URL dell'ACS a cui mandare la risposta.

Il passo 6 invece contiene (sempre in modo opaco):

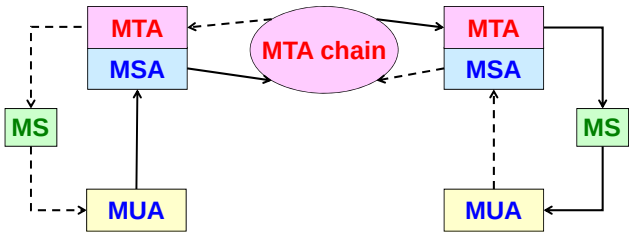
- La conferma della URL del servizio Google richiesto dall'utente.
- La risposta SAML di autenticazione con firma XML.
- La URL dell'ACS a cui la risposta è destinata.

Capitolo 9

Sicurezza della posta elettronica

La posta elettronica ha la peculiarità di essere l'unico servizio di largo utilizzo in cui la comunicazione non è realmente end-to-end. In particolare la posta elettronica funziona tramite il cosiddetto *Message Handling System (MHS)*, in cui entrano in gioco 4 IP di attori diversi:

- *Message User Agent (MUA).*
- *Message Submission Agent (MSA).*
- *Message Transfer Agent (MTA).*
- *Message Store (MS).*



Nel momento in cui si vuole inviare della posta in realtà si sta utilizzando un MUA, il quale permette di comporre il messaggio di posta elettronica.

Per la spedizione il MUA si basa su un server chiamato MSA, ossia quello che conosce come funziona il servizio di posta elettronica mondiale ed è in grado di introdurre il messaggio per il trasporto.

Il messaggio viene trasportato da una catena di MTA, ossia dei server speciali che sono in grado di trasferire la posta da una parte all'altra della rete Internet. L'ultimo MTA in uscita dalla catena sarà quello più vicino al server dove la posta viene depositata per il destinatario; questo server prende il nome di MS.

A sua volta il destinatario userà un MUA non per comporre il messaggio ma per leggere i messaggi che vengono ricevuti.

Si noti che il sistema funziona in entrambi i versi.

Questo sistema è analogo a come viene distribuita la posta fisica: la MUA altro non è che il foglio di carta su cui viene scritto il testo, l'MSA è la cassetta delle lettere dove viene depositata la lettera, gli MTA sono i vari postini e mezzi di trasporto con i quali la lettera viene trasferita, il MS è la buca delle lettere del destinatario e la MUA sarà ad esempio la chiave con cui il destinatario apre la cassetta per prendere la lettera.

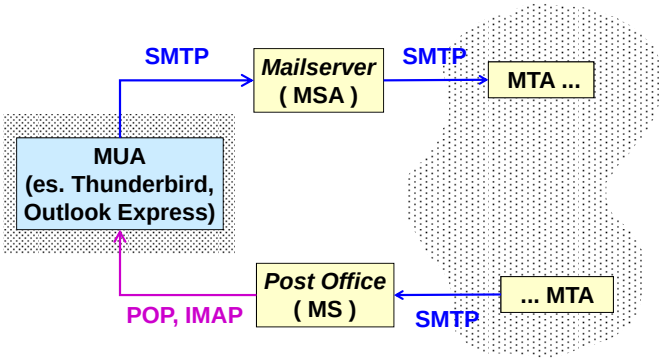
Da questo schema si può osservare che tra il MUA del mittente e quello del destinatario non esiste un collegamento diretto, ma questo ha senso visto che il servizio di posta elettronica è un servizio asincrono in cui i vari passi possono essere fatti in tempi diversi. Questa caratteristica

rappresenta una grande differenza rispetto agli altri sistemi di sicurezza visti finora, che invece funzionano in tempo reale.

9.1 Posta elettronica - Schemi implementativi

9.1.1 E-mail in client-server

Lo schema client-server è quello più classico e sicuro, anche se oggigiorno non è più molto in uso.



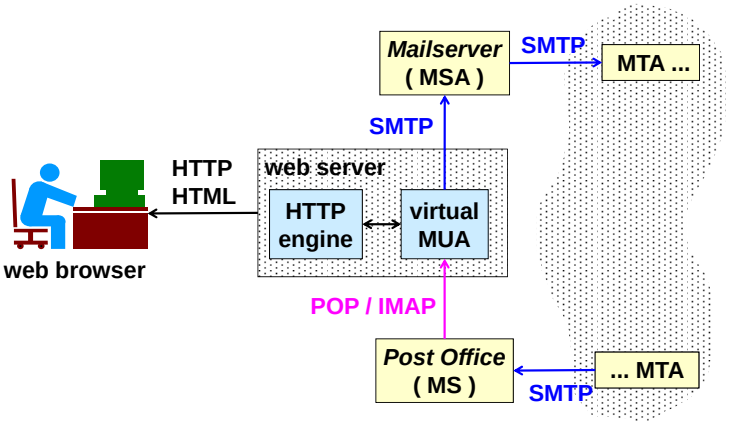
L'utente, dotato di un dispositivo di posta personale come ad esempio un PC, ha installato su di esso un software di MUA. Questo deve essere configurato correttamente in modo da sapere chi è il *Mailserver* (MSA), per poter inviare correttamente la posta, e il *Post Office* (MS), per poter leggere la posta ricevuta.

Solitamente nei software di MUA non si usa la nomenclatura MSA e MS, sono concetti troppo tecnici. È più probabile che il MSA sia indicato come Outgoing mailserver e il MS sia indicato come Incoming mailserver.

L'invio e la lettura della posta si basano sull'uso di protocolli diversi: per quanto riguarda l'invio si utilizza un protocollo di tipo push chiamato SMTP, che "spinge" la posta da un nodo all'altro della rete finché non giunge all'MTA di destinazione. Per la lettura dei messaggi invece si utilizzano dei protocolli di tipo pull, che permettono di "prelevare" la posta inserita all'interno della propria cassetta delle lettere; i protocolli usati per tale scopo sono il POP e l'IMAP.

9.1.2 Webmail

Poiché molto spesso gli utenti non hanno un loro dispositivo personale, sempre più spesso vengono utilizzate le webmail. Una webmail è un qualcosa che presenta un'interfaccia Web per il sistema di posta elettronica.



L'utente al posto di avere una MUA reale sul proprio device personale ha invece una MUA virtuale ospitata dal suo provider di posta elettronica. Per interagire con tale MUA si sfrutta un HTTP engine, il quale permette di usare un browser per fare le operazioni di spedizione e di ricezione posta.

In questo modo ci si rende indipendenti dal particolare dispositivo, nel senso che è possibile usare moltissimi browser per poter accedere alla proprio account di posta elettronica. Inoltre l'utente non dovrà più preoccuparsi di configurare correttamente il MSA e il MS, essi verranno configurati in modo opportuno dai gesteri del virtual MUA.

Dal punto di vista della sicurezza, quando l'utente legge la posta questa viene tolta dal Post Office e viene depositata sui server del provider della posta elettronica. In altre parole la posta non è propria dell'utente, ma è di colui che fornisce il servizio. Nello schema precedente una volta che l'utente ha tolto la posta dal Post Office, la posta risiede localmente sul device personale dell'utente; in altre parole essa è totalmente di sua proprietà.

I protocolli usati per le trasmissioni sono gli stessi dello schema precedente: SMTP e POP/IMAP.

9.2 Simple Mail Transfer Protocol (SMTP)

- *Simple Mail Transfer Protocol (SMTP)*: protocollo usato per mandare in push la posta. Utilizza la porta 25/TCP per parlare con gli MTA e la porta 587/TCP per parlare con l'MSA.
- *Post Office Protocol (POP)*: protocollo pull usato per leggere la posta, sfrutta la porta 110/TCP.
- *Internet Message Access Protocol (IMAP)*: protocollo sempre usato per leggere la posta, si basa sulla porta 143/TCP.
- *RFC*-822: formato del messaggio di posta elettronica. Un messaggio in formato RFC-822 è un messaggio di puro testo, ossia presenta solamente un body che è completamente intelligibile da un essere umano.
- *MIME*: estensione multimediale di RFC-822, per permettere la trasmissione di dati che non siano esclusivamente puro testo.

9.3 Messaggi RFC-822

In un messaggio RFC-822 presenta alcuni vincoli:

- È possibile usare solamente i caratteri dell'alfabeto americano a 7 bit (US-ASCII a 7 bit); l'ottavo bit è considerato non significativo, nonostante sia presente.
- Tutte le righe devono essere terminate con <CR> <LF>. Questo perché i vari SO utilizzano un terminatore di riga diverso: Windows utilizza <CR> <LF>, Unix usa <LF>, MAC usa <CR>.
- I messaggi sono composti da un header e da un body
  - Header: contiene delle parole chiavi ad inizio riga, e se una riga è più lunga di una determinata dimensione si prosegue sulla riga successiva; le righe di continuazione iniziano con uno spazio.
  - Body: separato dall'header da una riga completamente vuota, e contiene il messaggio vero e proprio.

9.3.1 Header RFC-822

Si noti la differenza fra questi due header:

- From: mittente logico del messaggio, può essere definito a piacere.
- Sender: è il mittente reale del messaggio, l'indirizzo dal quale il messaggio è stato realmente inviato.

In un messaggio, all'interno della sezione header, viene aggiunta una riga received per ogni MTA che viene attraversato; è un meccanismo simile al traceroute. In questo modo è possibile vedere la MTA chain.

From:	**mittente (logico)**
Sender:	**mittente (operativo)**
Organization:	**organizzazione del mittente**
To:	**destinatario**
Subject:	**argomento**
Date:	**data e ora di spedizione**
Received:	**passaggi intermedi**
Message-Id:	**ID di spedizione**
CC:	**in copia a**
Bcc:	**in copia (nascosta) a**
Return-Receipt-To:	**ricevuta di ritorno a**

9.3.2 Un esempio SMTP/RFC-822

```
telnet duke.colorado.edu 25
   Trying .....
   Connected to duke.colorado.edu
   Escape character is '^]'
   220 duke.colorado.edu ...
HELO leonardo.polito.it
   250 Hello leonardo.polito.it ... Nice to meet you!
MAIL FROM: cat
   250 cat ... Sender ok
RCPT TO: franz
   250 franz ... Recipient ok
DATA
   354 Enter mail, end with "." on a line by itself
From: cat@athena.polito.it (Antonio Lioy)
To: franz@duke.colorado.edu
Subject: vacanze

Ciao Francesco,
ti rinnovo l'invito a venirmi a trovare nelle tue
prossime vacanze in Italia. Fammi sapere
quando arrivi.

Antonio
.
   250 Ok
QUIT
   221 duke.colorado.edu closing connection
   connection closed by foreign host
```

1. Nell'esempio si usa il comando `telnet` per collegarsi al server di posta elettronica *duke.colorado.edu* tramite la porta *25*. Il server risponde con un codice positivo.
2. L'utente si presenta al server tramite il comando `HELO leonardo.polito.it`, al quale il server risponde nuovamente con un messaggio positivo.
3. `MAIL FROM: cat` serve a definire il mittente del messaggio. Anche in questo caso si riceve una risposta positiva dal server in quanto esso non può sapere l'identità reale dell'utente.
4. `RCPT TO: franz` definisce invece il destinatario del messaggio. Il server ci mette un po' a rispondere a tale comando perché deve controllare se il destinatario indicato esiste realmente all'interno di tutte le caselle di posta elettronica; in ogni caso in questo esempio il server risponde positivamente.
5. `DATA` sta ad indicare che l'utente sta per inserire il messaggio vero e proprio. Il server risponde con il codice 354, che sta ad indicare che se il messaggio sarà formattato correttamente verrà inviato. Una volta che il testo è stato inserito l'utente inserisce una riga contenente solamente "."

6. Il server controlla il messaggio, e poiché la sintassi è corretta risponde con un messaggio positivo.

7. `QUIT` serve infine per disconnettersi dal server.

Come si può facilmente capire dall'esempio questo è un sistema eccezionale per poter inviare delle mail false perché non si ha assolutamente nessun controllo per vedere l'identità della persona che ha inviato tale mail.

È infatti possibile inserire un qualunque indirizzo come mittente.

9.3.3 Problematiche

Sicurizzare la posta elettronica non è banale:

- Sistema connectionless, ossia non si ha un collegamento diretto tra il mittente e il destinatario. Anzi si parla di sistema *store-and-forward*, ossia il messaggio prima di essere inviato da un nodo all'altro viene prima salvato in locale e solo in un secondo momento viene inoltrato.
- Uso di MTA non fidati su cui si va a memorizzare (anche se temporaneamente) i messaggi.
- Livello di sicurezza del MS sul quale si va a memorizzare i messaggi prima che essi vengano letti.
- Si potrebbe implementare un sistema di cifratura, ma si avrebbero dei problemi nel caso in cui si usasse una mailing-list (Come implementare la cifratura per tutti gli appartenenti della mailing-list?).
- Poiché la posta elettronica è ormai un sistema di largo utilizzo, apportare delle modifiche non è una cosa così semplice da fare.

9.4 Mail spamming

Anche detto *Unsolicited Bulk E-mail* (UBE) o *Unsolicited Commercial E-mail* (UCE), il termine *spamming* è usato per indicare dei messaggi di posta elettronica indesiderati mandati per scopi commerciali, ma anche per scopi di attacco (da cui nasce la necessità di considerare questo argomento anche nel campo della sicurezza). In particolare le mail di spam possono contenere dei malware, possono essere utilizzate per manovrare un attacco di phishing, ecc.

Il problema principale legato allo spamming è che oggi rappresenta circa l'88% di tutto il traffico di posta elettronica. Come conseguenza tutta questa mole di dati causa un grosso carico sui server (impegnando CPU e memoria) e sulla rete (occupando la banda inutilmente), oltre ovviamente a generare un grosso fastidio per gli utenti.

La parola "spam" nasce da uno sketch dei Monty Pyton (gruppo comico britannico), in cui c'era della carne di maiale in scatola di bassissima qualità di marca spam. Da questa analogia si può capire il perché la posta buona viene invece chiamata "ham".

9.4.1 Strategie degli spammer

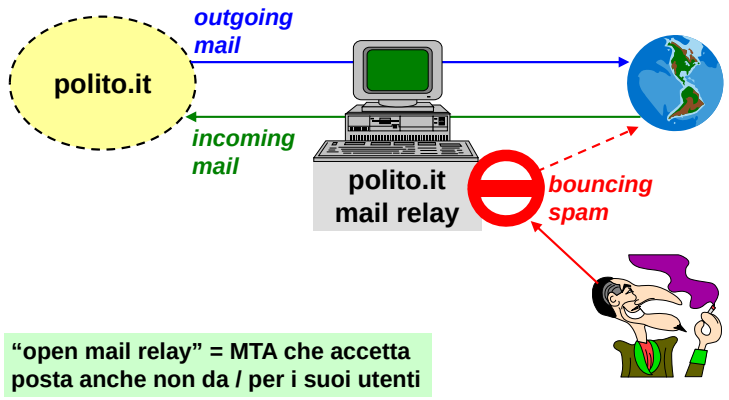
Gli spammer utilizzano diverse strategie per fare in modo che i propri messaggi siano accettati:

- Nascondere il vero mittente, andando comunque ad usare un mittente esistente. Ad esempio è comune mettere come mittente la stessa e-mail del destinatario.
- Spedire spam tramite degli MTA speciali
  - *Open mail relay*, che accettano di spedire posta anche per utenti che non fanno parte del loro dominio.

- Zombie o botnet, istruiti ad inviare non le e-mail del vero utente ma le e-mail per conto degli spammer.
- Con indirizzo IP variabile o inesistente, in modo da rendere difficile l'identificazione degli MTA usati dagli spammer.

- Visto che una delle strategie anti-spam è quella di andare a guardare il testo della e-mail in modo da controllare se contenga o meno della pubblicità, si cerca di rendere difficile l'identificazione del messaggio facendo *content obfuscation*, ossia cercare di rendere il messaggio non facilmente leggibile da programmi (ma facilmente comprensibile dagli esseri umani)
  - Errori deliberati (es. Viagra = Vi@gr@)
  - Inserire un'immagine al posto del testo, basandosi sul fatto che i programmi di riconoscimento automatico funzionano bene sul testo ma molto male sulle immagini.
  - Poiché una delle strategie anti-spam è quella di andare a vedere se un messaggio da controllare ha qualche similitudine con altri messaggi di spam tramite l'uso di tecniche di stima bayesiana, lo spammer esegue del *bayesian poisoning*, ossia inserisce il corpo della sua mail seguito da del testo casuale (es. frasi estratte da un libro). Questo serve solo per alterare la statistica delle parole.
  - Inserire il testo della mail di spam all'interno di una (finta) mail di errore, sfruttando il fatto che non tutti i sistemi anti-spam controllano i messaggi di errore.

(Open) mail relay



Il *mail relay* è tipicamente quel sistema che controlla la posta in ingresso e in uscita verso un determinato dominio.

Logicamente esso corrisponde all'MTA in uscita o in ingresso:

- Uscita: deve permettere alla posta generata all'interno del dominio di raggiungere Internet.
- Entrata: se qualcuno da Internet decide di spedire posta a quel dominio, il mail relay deve accettarla e trasferirla al dominio in questione.

Quello che un mail relay non dovrebbe essere in grado di fare è quello di accettare della posta proveniente dall'esterno, il cui mittente desidera che sia nuovamente inoltrata verso l'esterno. Poiché gli MTA sono sistemi di tipo store-and-forward questa operazione è teoricamente del tutto lecita.

La differenza tra un mail relay normale e un open mail relay sta nel fatto che il primo è configurato in modo che non possa essere usato per inoltrare mail ad utenti non appartenti al dominio gestito, mentre il secondo è mal configurato e può essere sfruttato dagli spammer per inviare delle mail per conto loro ad utenti esterni al dominio.

9.4.2 Strategie anti-spam per MSA

Una prima strategia è quella di cercare di evitare che gli spammer iniettino la posta nel circuito, andando a definire delle regole che riguardino gli MSA:

- Non configurare mai i propri MSA/MTA come open relay, ma restringerne l'uso solo ai propri utenti.
- Legata alla strategia precedente nasce il problema di come identificare gli utenti. In particolare l'autenticazione potrebbe essere fatta in diversi modi:
  - Indirizzo IP del MUA, ma in questo caso si avrebbe il problema dei nodi mobili, dell'esistenza dell'IP spoofing e della possibilità di avere un malware installato su un nodo valido.
  - Valore del campo From, ma questo è facilmente aggirabile tramite delle fake mail.
  - Autenticazione del MUA a livello di SMTP.

9.4.3 Strategie anti-spam per incoming MTA

Un sistema di anti-spam può anche essere messo sull'incoming MTA, ossia quello che controlla la posta in ingresso per un determinato dominio. In particolare l'incoming MTA dovrà decidere se rifiutare o accettare mail da un MTA esterno andando a controllare una blacklist o una whitelist. Questo può essere fatto tramite diverse soluzioni.

DNS-based BlackList (DNSBL)

Lo schema DNSBL è stato normato nell'RFC-5782 "DNS blacklists and whitelists". Il suo funzionamento è il seguente:

- Si riceve una richiesta di collegamento da parte di un MTA esterno, avente indirizzo A.B.C.D.
- La domanda che bisogna porsi è se questo indirizzo è noto per spedire spam. Questo viene verificato eseguendo un'operazione di lookup: nslookup –q=A D.C.B.A.dnsbl.antispam.net (esistono diversi domini DNS che forniscono un servizio di anti-spam: MAPS RBL, SORBS, ecc.). Se si riceve come risposta:
  - NXDOMAIN (No such domain), allora l'MTA non è uno spammer.
  - Un indirizzo, allora l'MTA sarà uno spammer. In particolare gli indirizzi che vengono forniti sono tutti del tipo 127.0.0.X, dove X è un codice che indica perché quel nodo è stato identificato essere uno spammer.
    Inoltre se si esegue una query lookup di tipo TXT, si ricevono maggiori informazioni.

Si noti che una volta che si viene inseriti in una lista DNSBL non è banale uscirne: occorre configurare il proprio MTA bene fin dall'inizio.

A tale proposito l'RFC-2142 ha introdotto l'obbligo di creare per ciascun dominio una casella di posta *abuse*@dominio, alla quale le persone esterne possono mandare delle mail per segnalare un abuso nell'uso del servizio di posta elettronica da parte di quel dominio. Questo risulta conveniente per il gestore del dominio, in quanto gli permette di effettuare delle verifiche e di prendere eventuali decisioni prima che il dominio venga inserito all'interno di una blacklist.

URI DNSBL

Poiché lo schema precedente si basa sul fatto che un certo indirizzo sia noto per inviare spam, esso non va bene nel caso in cui lo spammer cambi continuamente l'indirizzo mittente dei suoi messaggi. Per tale ragione un'altra tecnica che viene spesso usata è quella di andare ad accettare comunque tutte le mail, ma prima di inviarle nella casella postale del destinatario si esegue un

controllo sul loro body andando a vedere se contengono delle URI. Questo schema viene definito *URI DNSBL*, ed è basato sull'andare a definire la reputazione delle URI contenute all'interno del corpo delle mail.

Questa schema viene spesso implementato usando delle honeypot, che in questo caso prendono il nome di *spamtrap*. Questi altro non sono che dei domini che accettano mail di qualunque genere al solo fine di classificarle in base ad eventuali URI che contengono.

Greylisting

Greylisting è un altro metodo per difendere gli utenti dallo spam via e-mail. Un mail server che utilizza questa tecnica rifiuterà temporaneamente tutte le e-mail da un mittente che non conosce. Se l'e-mail è legittima, il mail server del mittente ritenterà l'invio, e questa volta l'operazione verrà accettata. Se l'e-mail viene inviata da uno spammer, probabilmente non si avrà un ulteriore tentativo poiché il mail server dello spammer avendo a disposizione migliaia (o milioni) di indirizzi email, passerebbe oltre non occupandosi degli errori.

Questa tecnica ha il grosso svantaggio di ritardare non soltanto le mail di spam, ma anche le ham: qualunque messaggio di posta elettronica che si riceve verrà ritardato di un certo delta (solitamente pari a 5 minuti). Inoltre l'incoming MTA ha anche un carico maggiore del normale perché deve tenersi una tabella di tutti gli MTA che lo hanno contattato negli ultimi minuti, in modo da sapere quali sono quelli che hanno già tentato un altro invio.

Nolisting (Poor man's greylisting

A causa di un maggior carico di lavoro richiesto ai server, è stata anche sviluppata la tecnica del *nolisting*. Essa è sempre basata sulla teoria che gli spammer non hanno tempo da perdere, e quindi non vanno a provare tutti gli MX (Mail eXchanger, ossia un'incoming MTA nella terminologia dei DNS) definiti per ciascun dominio. In particolare i domini che utilizzano questa tecnica vanno a definire tre diversi MX:

- 1° MX: non risponde mai.
- 2° MX: funziona regolarmente.
- 3° MX: non risponde mai.

Questo viene fatto perché solitamente gli spammer utilizzano il primo o il terzo MX, non li provano tutti.

Questa tecnica ha il vantaggio di bloccare, o comunque ritardare, gli spammer ma causa anche un ritardo dell'ham. Si ha però il vantaggio di non sovraccaricare ulteriormente l'MTA perché si andrà a ricevere solamente la posta valida e non è necessario tenere traccia di tutti gli MTA esterni che lo hanno contattato.

DomainKeys Identified Mail (DKIM)

Tecnica anti-spam definita in diversi RFC, è basata sull'uso di una whitelist, ossia una lista contenente un elenco di MTA che ufficialmente sono stati delegati a mandare posta per un certo dominio.

L'outgoing MTA di un determinato dominio di posta deve garantire crittograficamente

- L'identità del mittente.
- L'integrità (parziale) del messaggio. Questo viene fatto per evitare che un messaggio valido possa essere modificato da un MTA intermedio, facendolo diventare un messaggio di spam.

Queste caratteristiche vengono garantite tramite l'uso di una firma digitale. Tale firma:

- Viene apposta dall'ougoing MTA oppure direttamente dall'MSA.
- Copre alcuni degli header RFC-822 e una parte del body.

- Deve essere verificabile tramite l'uso di una chiave pubblica. Solitamente quello che viene fatto è quello di mettere le chiavi pubbliche all'interno del DNS.

Questa tecnica viene usata in modo sempre più crescente: ad esempio viene utilizzata da Gmail e da Yahoo per identificare la posta che è stata inviata da uno dei loro MTA.

Permette di scartare messaggi con un falso mittente, e quindi svolge una funzione di anti-spam e in parte di anti-phishing, nel senso che non si può mandare messaggi a nome di un'altra persona.

Sender Policy Framework (SPF)

Strategia di anti-spam definita nell'RFC-4408, è basata in un certo senso sempre sulla crittografia.

Gli incoming MTA sono già dichiarati nel DNS tramite gli MX record, ma non si ha nessuna informazione su quali siano gli outgoing MTA di quel dominio. A tale proposito è stato aggiunto un apposito record nel DNS per dichiarare quali sono gli outgoing MTA validi.

```
$ nslookup -q=txt polito.it.
polito.it text = "v=spf1 ptr ~all"
$ nslookup -q=txt gmail.com.
gmail.com text = "v=spf1 redirect=_spf.google.com"
$ nslookup -q=txt _spf.google.com.
_spf.google.com text = "v=spf1 ip4:216.239.32.0/19
   ip4:64.233.160.0/19 ip4:66.249.80.0/20 ip4:72.14.192.0/18
   ip4:209.85.128.0/17 ip4:66.102.0.0/20 ip4:74.125.0.0/16
   ip4:64.18.0.0/20 ip4:207.126.144.0/20 ip4:173.194.0.0/16
   ?all"
```

- 1° esempio: qualunque calcolatore all'interno del Poli che abbia un record ptr valido, può mandare posta elettronica in uscita.
- 2° esempio: elenco di tutti quanti gli indirizzi IPv4 degli outgoing MTA validi per Google.

9.5 Extended SMTP (ESMTP)

Un'altra strategia usata per bloccare lo spam è quella di impedire che lo spam venga iniettato nel sistema; questo deve essere fatto autenticando sull'MSA i MUA che intendono spedire posta. Tuttavia nell'SMTP base non ci sono strategie di questo genere.

A tale proposito si è deciso di estendere il protocollo, portando alla nascita del protocollo ESMTP. Questo protocollo non cambia il protocollo di base e nemmeno la modalità di trasmissione sul canale. Quello che viene cambiato è il messaggio di saluto HELLO con

EHLO hostname

Se il server ricevente parla anche lui ESMTP, allora esso dovrà dichiarare le estensioni che supporta, una per riga, nella sua risposta; in caso contrario risponderà dicendo messaggio sconosciuto.

9.5.1 Esempi ESMTP positivi

Mailer ESMTP senza estensioni

```
   220 mail.polito.it - SMTP service ready
EHLO mailer.x.com
   250 Hello mailer.x.com - nice to meet you!
```

1. Si esegue una richiesta di collegamento al server del Politecnico.
2. Si dichiara di voler usare la versione estesa del protocollo inviando il messaggio di EHLO.

3. Il server risponde con un messaggio positivo (250), che sta indicare che anche lui è in grado di parlare ESMTP. Tuttavia a questo messaggio non contiene nessuna estensione.

Mailer ESMTP con estensioni

```
   220 mail.polito.it - SMTP service ready
EHLO mailer.x.com
   250-Hello mailer.x.com - nice to meet you!
   250-SIZE 26214400
   250 8BITMIME
```

1. Si esegue una richiesta di collegamento al server del Politecnico.
2. Si dichiara di voler usare la versione estesa del protocollo inviando il messaggio di EHLO.
3. Il server risponde con un messaggio positivo; il codice 250- sta ad indicare che la riga corrente sarà seguita da un'altra. In particolare nelle due righe successive sono indicate le estensioni supportate:
   - *SIZE*: estensione importante perché dichiara qual è la massima lunghezza di un messaggio di posta che il server andrà ad accettare.
   - 8BITMIME: estensione che permette di sapere che il server nella ricezione della posta non va a modificare l'ottavo bit; in alte parole è possibile inviare dei dati che sono significati anche sull'ottavo bit.

9.5.2 Esempio ESMTP negativo

```
   220 mail.polito.it - SMTP service ready
EHLO mailer.x.com
   500 Command not recognized: EHLO
```

1. Si esegue una richiesta di collegamento al server del Politecnico.
2. Si dichiara di voler usare la versione estesa del protocollo inviando il messaggio di EHLO.
3. Il server risponde con un messaggio negativo (500), che sta a significare che esso è in grado di parlare solamente la versione base del protocollo.

9.5.3 SMTP-Auth

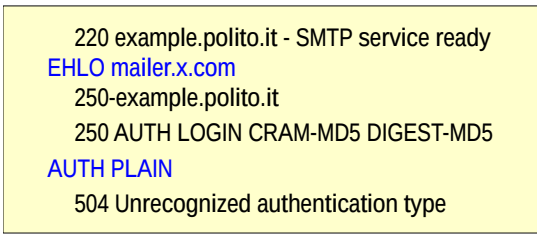
Tra le varie estensioni che sono state definite per l'ESMTP, esiste l'estensione *SMTP-Auth* (RFC-4954) usata per fare l'autenticazione.

Questa estensione introduce il comando AUTH più alcune opzioni del comando MAIL FROM.

In generale comunque lo scopo è quello di autenticare un client (MUA) prima di accettarne i messaggi. Questo è molto utile come strategia anti-spamming:

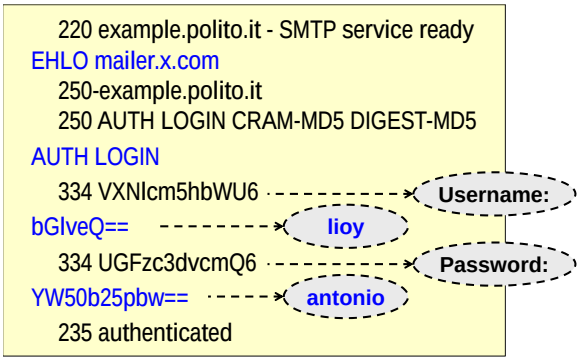
- Dopo il comando EHLO il server invia i meccanismi di autenticazione supportati.
- Il client ne sceglie uno.
- Viene eseguito il protocollo di autenticazione.
- Se l'autenticazione fallisce, il canale viene chiuso. In caso invece l'autenticazione vada a buon fine si ha la certezza che si tratti di un MUA valido che ha diritto ad usare l'MTA per spedire posta.

Esempio AUTH negativo



1. Si esegue una richiesta di collegamento al server del Politecnico.
2. Si dichiara di voler usare la versione estesa del protocollo inviando il messaggio di EHLO.
3. Il server risponde con un messaggio positivo (250), e in particolare dichiara di supportare l'autenticazione secondo i metodi LOGIN, CRAM-MD5 e DIGEST-MD5.
4. Il client dovrebbe scegliere tra uno dei metodi di autenticazione proposti dal server, ma non supportandone nessuno cerca di eseguire l'autenticazione usando il metodo PLAIN.
5. Il server risponderà con un messaggio di errore, e quindi l'autenticazione fallirà.

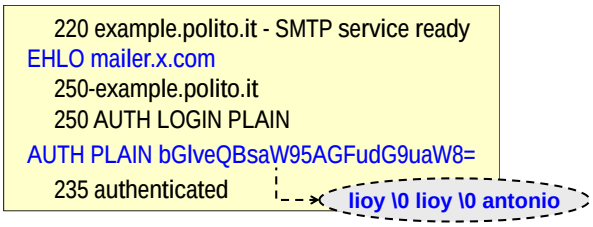
AUTH: Metodo LOGIN



1. Si esegue una richiesta di collegamento al server del Politecnico.
2. Si dichiara di voler usare la versione estesa del protocollo inviando il messaggio di EHLO.
3. Il server risponde con un messaggio positivo (250), e in particolare dichiara di supportare l'autenticazione secondo i metodi LOGIN, CRAM-MD5 e DIGEST-MD5.
4. Il client sceglie di autenticarsi usando il metodo LOGIN.
5. Ad una prima analisi può sembrare che le righe che seguono non siano intellegibili. Ma bisogna ricordarsi che la posta elettronica standard usa un canale a 7 bit, e di conseguenza quello che è stato fatto è mappare tutti i dati trasferiti su 7 bit; in particolare si va ad usare la codifica in BASE64.
6. Il server controllando che i dati inseriti dall'utente siano corretti, ne esegue l'autenticazione.

Questo metodo non è particolarmente sicuro, perché chiunque stia sniffando la rete deve semplicmente eseguire una decodifica in base 64 dei dati per ottenere username e password degli utenti.

AUTH: Metodo PLAIN



Variante del metodo LOGIN che evita di fare 4 scambi di messaggi perché username e password vengono mandati direttamente quando si esegue il comando AUTH PLAIN, tutto sempre codificato in BASE64.
AUTH PLAIN id_pwd_{BASE64}
La stringa id_pwd è definita come

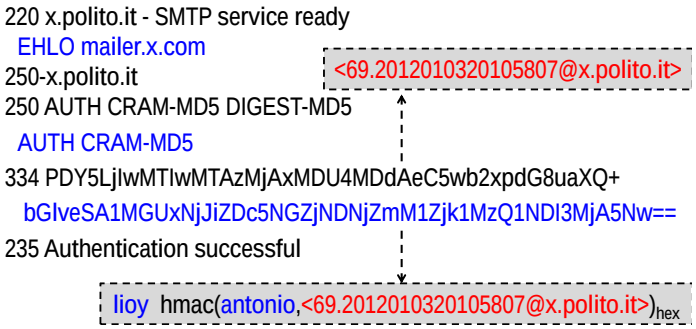
$$[\ authorize\_id\ ]\ \backslash 0\ authentication\_id\ \backslash 0\ pwd$$

- *[authorize_id]*: identificativo di autorizzazione (Opzionale).
- *authentication_id*: Identificato di autenticazione del client.
- *pwd*: password del client.

A parte rendere la procedura di autenticazione più veloce, questo metodo non introduce nessun livello di sicurezza ulteriore rispetto al metodo LOGIN. Questo significa che LOGIN e MAIN sarebbero sconsigliati a meno che non vengano usati su canali sicuri.

AUTH: Metodo CRAM-MD5

Challenge-Response Authentication Mechanism (CRAM) è un metodo di autenticazione basato sull'algoritmo HMAC-MD5.



Il protocollo è essenzialmente composto da tre fasi:

- *Challenge*: nell'autenticazione CRAM-MD5, il server invia una stringa di sfida al client.
- *Response*: il client risponde con una stringa con questa struttura:
  - La stringa inviata dal server, codificata con BASE64, viene decodificata.
  - La stringa decodificata viene criptata con HMAC-MD5 usando la password dell'utente come chiave segreta.
  - La sfida codificata viene convertita in cifre esadecimali.
  - La username ed uno spazio blank vengono aggiunti all'inizio della stringa hex
  - La concatenazione viene codificata in BASE64 ed inviata al server

- *Comparazione*: il server, conoscendo la password associata all'utente, esegue lo stesso metodo del client e se la stringa calcolata è uguale a quella inviata dal client allora l'autenticazione è concessa.

9.5.4 Protezione di SMTP con TLS

Nel caso in cui si volessero usare i metodi LOGIN e PLAIN invece che CRAM-MD5 occorre operare su un canale sicuro. Ma anche nel caso in cui si usasse CRAM-MD5 conviene utilizzare una canale sicuro, perché questo metodo risolve il problema dell'autenticazione ma non risolve il problema che qualcuno possa sniffare il messaggio durante il transito, andando a leggerne il contenuto e magari anche alterarlo.

A tale proposito si è deciso di applicare TLS al protocollo SMTP (RFC-2487). In particolare è stato aggiunto un nuovo comando chiamato *STARTTLS*, che è anche un'opzione di EHLO.

Se la negoziazione ha successo si resetta lo stato del protocollo; in altre parole si riparte dal messaggio di EHLO, e le estensioni supportate possono essere diverse.

Nel caso in cui invece il livello di sicurezza negoziato fosse ritenuto insufficiente:

- Se è il client a ritenerlo, allora esso andrà ad inviare il comando QUIT ed esce.
- Se è invece il server a ritenerlo, allora esso andrà a rispondere ad ogni comando ricevuto dal client con l'errore 554 (refused due to low security). Si lascia quindi sempre al client l'onere di chiudere il canale.

```
    220 example.polito.it - SMTP service ready
EHLO mailer.x.com
    250-example.polito.it
    250-8BITMIME
    250-STARTTLS
    250 DSN
STARTTLS
    220 Go ahead
... TLS negotiation is started between client and server
```

Si noti che il canale TLS potrebbe essere negoziato tra un MUA e un MSA, ma anche tra due MTA. Questo significa che questa soluzione non fornisce una protezione end-to-end ma hop-by-hop. Come conseguenza si potrebbe avere un canale sicuro su una tratta, e non sulla tratta successiva. In altre parole l'uso di questa estensione non fornisce nessuna garanzia sul fatto che tutto il percorso fatto da un messaggio di posta elettronica sia protetto.

Per questo motivo questa estensione non viene molto usata. Viene tipicamente usata per creare un canale sicuro tra un MUA e un MSA nel caso in cui si volesse usare il metodo LOGIN o PLAIN, o nel caso in cui si volesse cercare evitare lo sniffing almeno all'interno della rete locale.

9.6 Servizi di sicurezza per messaggi di e-mail

Poiché l'uso di TLS non garantisce una protezione end-to-end, occorre usare altri meccanismi di sicurezza che proteggano il messaggio intrinsecamente: è il messaggio ad essere end-to-end, il suo trasporto viene fatto hop-by-hop.

I servizi necessari per rendere la posta elettronica veramente sicura sono:

- *Integrità*, anche se non si ha una comunicazione diretta tra mittente e destinatario. In questo modo il messaggio non potrà essere modificato né durante la trasmissione né sui server intermedi.

- *Autenticazione*, per poter identificare con certezza il mittente e quindi evitare le fake mail.
- *Non ripudio*, in modo che il mittente non possa negare di aver spedito un messaggio.
- *Riservatezza* (opzionale), in modo che i messaggi non siano leggibili sia in transito sia nella casella postale del destinatario.

9.6.1 Sicurezza delle e-mail - Idee guida

- *Nessuna modifica agli attuali MTA*: è essenziale codificare i messaggi in modo da evitare problemi nell'attraversare i gateway.
- *Nessuna modifica agli attuali UA*: possibilità di avere delle interfacce utenti scomode.
- *Modifica agli attuali UA*: interfaccia utente migliore in quanto andrà ad integrare tutte quante le funzioni.

I maggiori MUA oggi esistenti (Outlook, Thunderbolt, ecc.) hanno già direttamente implementato tutte le funzioni che servono per fare posta elettronica sicura, e di conseguenza l'interfaccia utente è molto semplice da usare.

Dal punto di vista tecnico, gli strumenti usati per fare sicurezza delle mail sono:

- *Algoritmi simmetrici*, usati per fare riservatezza. In questo modo sarà possibile cifrare il messaggio (anche nell'ipotesi che abbia dimensioni molto grandi) con una chiave di messaggio.
- *Algoritmi asimmetrici*, usati per cifrare e scambiare con il destinatario la chiave simmetrica scelta, ma anche per la parte di firma digitale.
- *Certificati a chiave pubblica* (es. X.509), per supportare la funzione di non-ripudio.

In questo modo si avrà che la sicurezza del messaggio si baserà solo sulla sulla sicurezza dell'UA, e non su quella degli MTA (non fidati).

9.6.2 Tipi di messaggi sicuri

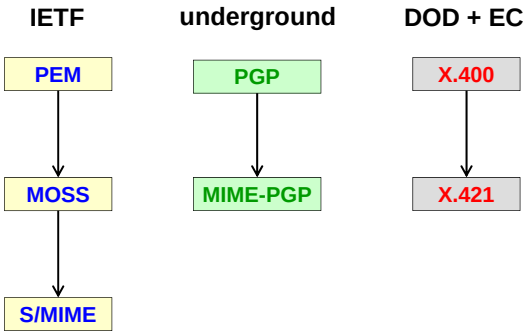
Esistono 4 tipi di messaggi diversi:

- *Clear-signed*: messaggio in chiaro (e di conseguenza leggibile da chiunque) + firma digitale (che può essere posta come allegato o può anche essere parte integrante del messaggio). La firma sarà verificabile solo da chi ha un MUA sicuro, ossia un MUA che supporta il protocollo di sicurezza utilizzato.
  Questo è il messaggio di posta elettronica sicuro più generale che esiste perché il messaggio è leggibile da chiunque (non è necessario sapere se il destinatario supporta posta sicura o meno), ed è inoltre presente una firma digitale che garantisce integrità, autenticazione e non-ripudio.
- *SIgned*: messaggio in chiaro + firma, entrambi codificati usando ad esempio base64 o anche altri metodi. Per verificare la firma, ma anche solo per leggere un messaggio, è necessario effettuare la decodifica del messaggio; di conseguenza occorre avere un MUA sicuro, oppure occorrerà fare uno sforzo manuale (si salva il messaggio come testo e poi se ne esegue la decodifica con un opportuno programma).
- *Encrypted/enveloped*: messaggio cifrato + chiavi cifrate (con la chiave pubblica del destinatario), il tutto codificato (tipicamente in base64). Solo chi ha un MUA sicuro e ha le chiavi crittografiche necessarie potrà andare a decifrare il messaggio.
- *Signed and enveloped*: messaggio + chiavi cifrate, il tutto codificato.

Messaggi sicuri - Creazione

- *Canonicalizzazione*: operazione non strettamente legata alla sicurezza, consiste nel mettere il messaggio in un formato standard, che sia indipendente dal OS, dal particolare host, dalla rete, ecc.
- *Message Integrity Code (MIC)*: l'aggiunta di questo codice garantisce integrità, autenticità e non-ripudio. Tipicamente questo codice viene creato concatenando il messaggio con un hash del messaggio stesso cifrato con la chiave privata del mittente.
$$msg + c(K_{pri\_sender}, h(msg))$$
- *Cifratura*: questo è fatto nel caso in cui si volesse avere anche riservatezza. Tipicamente la cifratura è fatta nel modo seguente:
$$c(K_m, msg) + c(K_{pri\_sender}, K_m)$$
  - K_m = chiave di messaggio generata al volo.
  - $K_{\text{pri\_sender}}$ = chiave pubblica del ricevente.
- *Codifica*: operazione eseguita in modo da evitare alterazioni da parte degli MTA intermedi. Tipicamente la codifica usata è base64 (in passato si usava anche lo uuencode e binhex).

9.7 Formati di posta elettronica sicura



Inizialmente l'IETF è andata a definire il formato *PEM*, che serviva a fare sicurezza sui messaggi di posta RFC-822 base.

Successivamente in seguito alla comparsa di MIME, che permetteva di avere gli allegati, è stato sviluppato *MOSS*, che altro non è che un PEM applicabile anche agli allegati. MOSS però non ha avuto molto successo perché da un punto di vista commerciale la RSA ha proposto un formato alternativo, simile come concetto ma diverso nell'implementazione, chiamato *S/MIME*. Questo oggi è lo standard de facto per la sicurezza della posta elettronica.

Il mondo underground però non ha accettato questi standard, principalmente per il fatto che essi richiedono l'uso di certificati X.509. E' stato allora sviluppato un formato alternativo chiamato *PGP*, seguito da *MIME-PGP* in modo da supportare MIME.

Il mondo militare (Department of Defense, DOD) e in parte anche le istituzioni europee per molto tempo hanno supportato *X.400*, e successivamente la versione *X.421* per il supporto alla multimedialità introdotto da MIME.

9.7.1 Pretty Good Privacy (PGP)

PGP è un sistema per fare autenticazione, integrità e riservatezza di dati generici. Originariamente è stato applicato a messaggi di posta elettronica, ma anche a file privati. I suoi obiettivi erano gli stessi di quelli di PEM, e anche la struttura era molto simile ma era molto più artigianale.

Questo era dovuto al fatto che PGP è stato sviluppato da un'unica persona, Phil Zimmerman, allo scopo di proteggere lo scambio di messaggi tra amici e l'upload di file. PGP è stato sviluppato in un'epoca in cui Internet era molto diversa da quella odierna. Per scambiarsi dei file si ricorreva alle cosiddette BBS (Bulletin Board System), ossia dei server a cui ci si collegava tramite modem, e in cui la gente metteva dei file che erano a tutti gli effetti dei messaggi. Questi messaggi potevano essere pubblici o privati, ossia destinati soltanto ad alcuni utenti della BBS. Il problema stava nel fatto che i gestori della BBS avrebbero potuto leggere i messaggi e modificarli. In questo scenario Zimmermann ha sviluppato un sistema di protezione che forniva come minimo integrità e autenticazione, in modo da evitatre modifiche da parte dei gestori, ed eventualmente anche riservatezza, in modo che i messaggi fossero leggibili solamente da parte di certi utenti. Questo sistema di protezione si poteva applicare anche alla posta elettronica in quanto concettualmente i sistemi erano molto simili. PGP è stato definito inizialmente nell'RFC-1991, il quale contiene le informazioni sullo schema scelto da Zimmermann. Più recentemente PGP è stato definito nell'RFC-4880, con la definizione dello standard OpenPGP.

Una delle particolarità di PGP è quella di non usare certificati X.509, ma di usare uno schema di certificazione delle chiavi molto peculiare in cui si parla di amici (fidati e non fidati), e di usare un algebra di propagazione della fiducia. PGP è famoso perché è il sistema più diffuso al mondo, sviluppato per tutte le piattaforme. PGP e Zimmermann sono diventati per lungo tempo un simbolo della libertà in Internet contro il tentativo dei governi di controllare Internet stessa.

Phil Zimmermann

Phil Zimmermann rilascia PGP come freeware nel 1991. I problemi sono iniziati nel momento in cui una copia del programma è stata caricata su un server in Finlandia. Poiché in quegli anni vigeva il divieto di esportazione di materiale crittografico, il Governo Americano ha incarcerato Zimmermann.

Zimmermann si è fin da subito dichiarato innocente, e la querela è andata avanti per diverso tempo (Zimmermann ha ricevuto supporto da molte persone, che facevano anche delle donazioni affinchè riuscisse a pagarsi gli avvocati) fino a quando nel 1996 l'avvocato del Governo Americano ha dichiarato di rinunciare a perseguitarlo. In seguito a questa vicenda, e fiero di aver vinto la sua causa, Zimmermann decide di fondare la PGP Inc., la quale verrà poi acquisita dalla NAI.

In seguito all'acquisizione PGP ha subito una serie di modifiche; Zimmerman continuava ad essere in un certo senso il garante che il sistema continuasse ad essere open, gratis per coloro che lo volevano usare per fini personali e protetto. Poiché il codice di PGP era opensource, è però stato scoperto che erano state aggiunte delle backdoor nel codice, che permettevano di decifrare i messaggi senza il consenso dell'utente. In seguito a questo si è venuta a creare una grossa polemica su Internet, e le dichiarazioni di Zimmermann sull'essere del tutto estraneo ai fatti non sono state accolte bene in quanto essendo lui il Chief Scientist che autorizzava tutte le modifiche qualche responsabilità le aveva sicuramente. Questo ha causato la totale perdità di fiducia nei suoi confronti.

Zimmermann ha allora deciso di fondare una nuova azienda nell'agosto del 2002, la PGP Co., la quale produce PGP ma soltanto per piattaforme Windows e inoltre è necessario pagare una licenza per utilizzarlo.

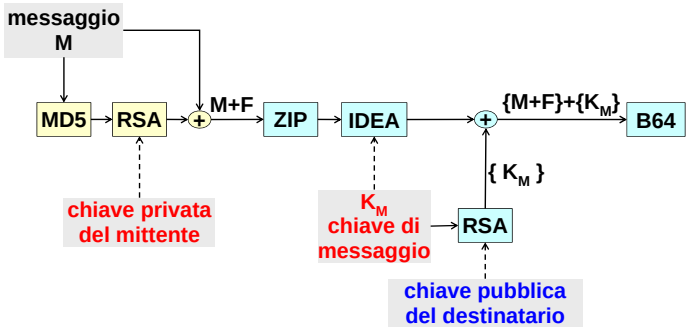
PGP - Algoritmi (Fino alla v. 2.6)

Fino alla v. 2.6 (versione originale di Zimmermann) PGP ha una struttura estremamente semplice, con l'uso di algoritmi fissi:

- IDEA per la cifratura simmetrica.
- MD5 per il digest.
- RSA per la parte asimmetrica, sia per la firma digitale sia per lo scambio della chiave simmetrica.

Queste scelte implementative erano state fatte perché tutti questi algoritmi sono gratuiti per usi non commerciali.

Esempio PGP 2.6 - Firma + Cifratura



1. Calcolato del digest MD5 del messaggio, il quale veniva poi cifrato RSA con la chiave privata del mittente.
2. Concatenazione del messaggio con firma calcolata al punto precedente.
3. ZIP dei dati per ridurne la dimensione ed eliminare le ridondanze.
4. Dati cifrati con IDEA usando una chiave di messaggio generata al volo.
5. Chiave di messaggio cifrata RSA con la chiave pubblica del destinatario.
6. Concatenazione dei dati genrati ai punti 4 e 5, e successiva codifica base64.

PGP - Certificazione delle chiavi

Il certificato altro non è che la chiave pubblica della persona che si sta certificando, firmata da tutte le persone che si fidano di questa persona. Non si ha quindi un'unica firma fatta dalla CA, ma ci sono tante firme fatte da tutti gli "amici" di questa persona.

La fiducia si propaga transitivamente con un certo grado di approssimazione. In altre parole significa che occorre fare una classificazione degli amici secondo una serie di categorie:

- *Completely*: amici di cui ci si fida completamente.
- *Partially*: amici di cui ci si fida parzialmente.
- *Untrusted*: amici di cui non ci si fida.
- *Unknown*: resto del mondo.

Lo schema nell'esempio 9.2 prende il nome di *web of trust*. Questo schema assomiglia al modo con cui noi ci fidiamo delle relazioni sociali e umane. Non è così banale revocare la fiducia nel momento in cui una chiave privata venisse compromessa.

Nell'esempio si ha che C è un amico di B, di cui esso si fida completamente. Poiché io utente mi fido completamente di B, per la proprietà transitiva della fiducia anche io mi fiderò di C (anche se per me è uno sconosciuto).

È necessario avere almeno 3 fiduce parziali affinchè la fiducia si propaghi. L'utente I ha la chiave firmata da F, G e H, e per questo motivo l'utente marcherà I come un utente parzialmente fidato.

M ha solo un utente in comune, F, con il quale si ha una fiducia parziale. Di conseguenza M non supera la soglia.

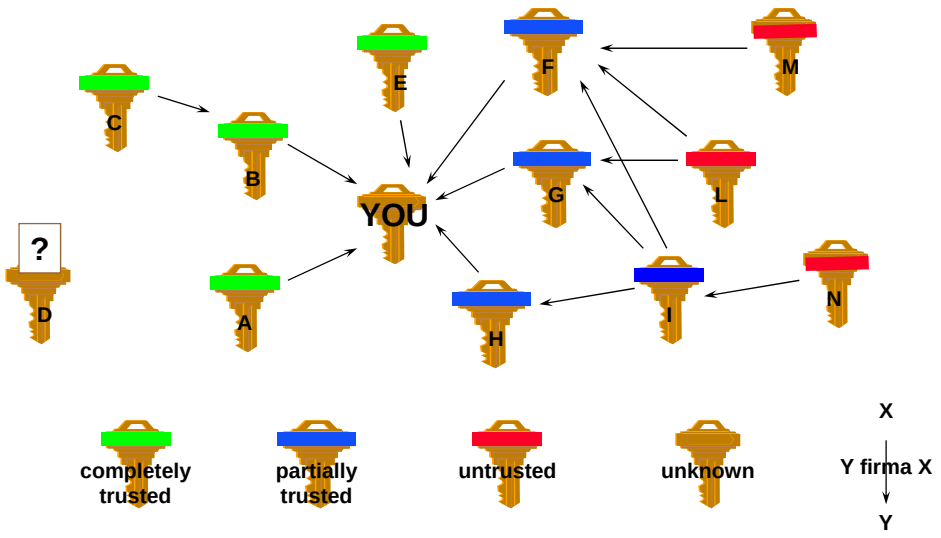


Figura 9.2: Web of trust.

PGP - Distribuzione delle chiavi

Le chiavi sono conservate personalmente da ogni utente nel cosiddetto key-ring. Esse sono distribuite direttamente dai proprietari, ad esempio all'interno dei famosi PGP party, oppure andando a caricarle all'interno di un key-server (che le distribuiva tramite http, smtp, finger, ...).

Ci sono stati anche alcuni progetti per distribuire le chiavi mediante X500 o DNS. Ad esempio esiste la rete pgp.net, su cui si può andare per scaricare le chiavi degli utenti.

PGP & NAI

PGP è stato acquisito da NAI nel dicembre del 1997.

Primo problema che NAI ha avuto è che gli algoritmi scelti da Zimmermann erano gratis per uso personale, ma a pagamento per uso commerciale. Di conseguenza è stato necessario sostituire gli algoritmi con altri che fossero gratuiti anche per usi commerciali; in particolare si è deciso di basare la firma su DSA, lo scambio chiavi su DH e la cifratura su 3DES.

PGP è stato integrato in molti sistemi di posta elettronica, e si è anche avuto qualche tentativo di introduzione nel mercato aziendale. Ma alle aziende non piaceva molto l'idea dei PGP party, e per tale ragione NAI ha suggerito di creare delle pseudo-CA nel senso di avere un super-firmatario (a posto di fare firmare la chiave da tutti gli amici la si faceva firmare dal responsabile aziendale della sicurezza). Questo però non ha avuto buon esito, e nel settembre del 1998 NAI ha deciso di sviluppare una versione che supportasse direttamente i certificati X.509; tuttavia questo non è mai stato fatto.

Nell'agosto del 2002 NAI ha ceduto tutti i diritti dul programma PGP alla PGP Co.

Gnu Privacy Guard (GPG)

Poichè PGP Co. ha deciso di non rilasciare più PGP in versione freeware, e soprattutto di svilupparlo solamente per la piattaforma Windows, la comunità di Internet si è sentita tradita e ha sviluppato una soluzione alternativa chiamata GPG.

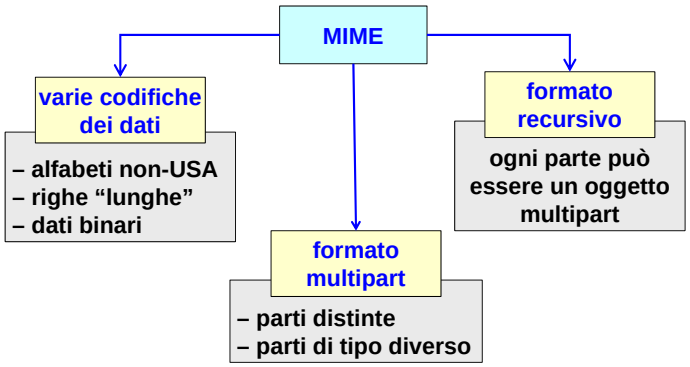
GPG è una riscrittura completa di PGP sotto licenza GPL e priva di qualunque algoritmo brevettato. Esso interopera sia con messaggi PGP 2.x e anche con i messaggi OpenPGP.

Si ha il supporto di tantissimi sistemi crittografici, numerosi front-end grafici ed è praticamente disponibile per qualunque piattaforma.

Si tenga presenta che PGP e GPG sono rimaste soluzioni di nicchia, usate da poche persone, ma sembra che qualcosa stia cambiando. Ad agosto del 2014 Gmail e Yahoo, a seguito dello

scandalo NSA, hanno dichiarato che integreranno dentro il loro sistema di web-mail OpenPGP; in questo momento questa soluzione è in corso di sviluppo. Questo è sicuramente una cosa positiva, perché permetterà di avere posta elettronica sicura anche per gli utenti web-mail, ma creerà sicuramente dei problemi di interoperabilità (S/MIME usa un formato completamente diverso ed è basato sui certificati X.509). Non è quindi chiaro se si avranno due sistemi diversi (S/MIME per sistemi corporate e PGP per gli utenti) o se si troveranno delle soluzioni per garantire la interoperabilità.

9.7.2 Multipurpose Internet Mail Extensions (MIME)



MIME è il supporto usato per estendere formati che tipicamente erano solo testuali in modo che possano supportare anche altri tipi di informazioni. In particolare MIME è nato come supplemento della posta elettronica proprio perché inizialmente la posta era puramente di tipo testuale.

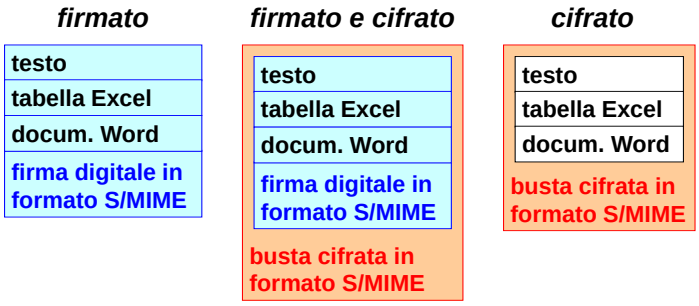
MIME offre essenzialmente 3 grandi innovazioni:

- *Varie codifiche di dati*
  - Possibilità di usare alfabeti diversi da quello USA a 7 bit. Questo è molto importante perché ad esempio se in un messaggio si vuole avere le lettere accentate occorre anche usare l'ottavo bit, che nell'RFC-822 era invece destinato ad altri scopi.
  - Possibilità di avere righe molto lunghe, mentre di base non si potevano avere righe più lunghe di 72 caratteri.
  - Possibilità di inserire in un messaggio dati binari generici, come immagini, video, ecc.
- *Introduzione del concetto di multipart*: un oggetto come il messaggio di posta elettronica può contenere al suo interno varie parti distinte, ognuna con un tipo diverso. Questo concetto è ad esempio utilizzato quando si invia un messaggio di testo con i cosiddetti allegati: in realtà non sono allegati, sono tutte parti distinte, tutte composte all'interno di un unico messaggio.
- *Formato recursivo*: ciascuna delle parti di un oggetto possono a loro volta essere degli oggetti MIME.

Tutti questi concetti sono in generale molto importanti per la posta elettronica, ma anche per quanto riguarda la sicurezza. Innanzitutto la possibilità di aggiungere dati binari è molto importante perché permette ad esempio di codificare all'interno di un messaggio la sua firma digitale. Inolre tipicamente la firma digitale è qualcosa di aggiunto al messaggio; ecco che allora avere un formato multipart permette di aggiungere la firma digitale come una parte del messaggio. Infine la recursione può essere utile nel caso di cifratura, nel senso che il messaggio MIME originale potrebbe essere contenuto in un altro oggetto MIME cifrato.

9.7.3 Posta elettronica multimediale sicura - MOSS o S/MIME

MIME è stato ampiamente sfruttato per fare la sicurezza della posta elettronica. In particolare sono stati creati due formati, MOSS e S/MIME, che offrono funzionalità di firma digitale e di cifratura dei messaggi MIME tramite l'utilizzo di certificati X.509v3.



Nell'immagine a sinistra si ha un generico messaggio in formato MOSS o S/MIME contenente alcune parti MIME e un'ultima parte corrispondente alla firma digitale (hash calcolato su tutte le parti MIME, cifrato con la chiave privata del mittente).

Nell'immagine a destra invece si ha un esempio di messaggio cifrato, in cui il messaggio originale MIME è stato inserito all'interno di un altro messaggio MIME dopo averne fatto la cifratura; in altre parole il messaggio cifrato è stato aggiunto come dati binari all'interno di un altro messaggio MIME.

Infine nell'immagine centrale si ha un esempio di messaggio firmato e cifrato, creato applicando le due tecniche precedenti.

9.7.4 RFC-1847 (MOSS)

Con l'RFC-1847 MOSS ha definito diverse estensioni per la sicurezza dei messaggi MIME. In particolare si era andato a definire sia la parte di firma digitale sia la parte di cifratura. Tuttavia la parte di cifratura oggi non è più usata, mentre per la firma digitale continua ad essere usata in parte con lo stesso schema.

La firma digitale viene fatta andando ad aggiungere al messaggio MIME un contenuto di tipo *multipart/signed*, ossia si dichiara che il messaggio è formato da diverse parti, di cui l'ultima è la firma digitale.

```
Content-Type: multipart/signed;
    protocol="TYPE/STYPE";
        micalg="...";
       boundary="..."
```

Si ha inoltre la dichiarazione della modalità con cui si va a generare la firma digitale (con l'indicazione di un tipo e sottotipo) e dell'algoritmo di MIC, ossia l'algoritmo di hash usato per calcolare il digest di tutte le parti precedenti la firma digitale. Infine si ha la dichiarazione del boundary, ossia dove termina una parte ed inizia quella successiva.

Vengono fatte due body part:

- Quella da proteggere;
- La firma, che ha come content type il tipo e il sottotipo dichiarato in precedenza.

Questo schema va in generale molto bene, ma ci potrebbero essere dei problemi se un gateway altera i messaggi. Questo è lo schema usato ancora oggi quando si vuole avere dei messaggi di tipo clear signed, ossia quelli leggibili a tutti ma che contengono la firma digitale.

9.7.5 S/MIME

S/MIME è il principale standard oggi usato per la sicurezza dei messaggi MIME.

Per un certo periodo di tempo MOS e S/MIME sono stati in competizione, nel senso che MOS era promosso dall'IETF mentre S/MIME era promosso da RSA (andava ad usare una serie di soluzioni proprietarie). La v2 di S/MIME è stata pubblicata solo come serie di informational RFC, senza quindi nessuna approvazione dall'IETF. Con il passare del tempo si è sempre più avuto il trend di muoversi verso S/MIME, il quale a partire dalla v3 è diventato un proposed standard IETF.

RFC-2634

In aggiunta ai servizi base di S/MIME, nell'*RFC-2634* vengono indicati anche dei servizi per offrire più sicurezza. In particolare ci sono 4 aree in cui si hanno avuto delle estensioni:

- *Firma per ricevuta di un documento*, in modo che il destinatario di un messaggio non possa negare di averlo ricevuto.
- *Security label*, meccanismo attraverso il quale si va a classificare ogni messaggio secondo una certa gerarchia di sicurezza. Ad esempio in ambito militare i messaggi vengono classificati in unclassified (leggibili da chiunque), classified (riservati solo a coloro che hanno una certa autorizzazione), secret e top-secret.
  Questo meccanismo viene in generale usato per trasmettere i messaggi usando solamente delle linee e degli MTA che abbiano almeno lo stesso livello di sicurezza. La security label comunque è solo un suggerimento, in quanto non è detto che l'infrastruttura abbia le capacità di basare il routing in base a questa caratteristica.
- *Mailing-list sicure*, che permette di risolvere il problema di come fare a cifrare i messaggi destinati ad una mailing-list (dove non si ha a disposizione le chiavi pubbliche dei destinatari). Questo però introduce una debolezza perché il messaggio non è più cifrato end-to-end ma il gestore della mailing list potrebbe potenzialmente andare a leggerne il contenuto.
- *Firma degli attributi dei certificati.*

Architettura di S/MIME

Da un punto di vista architetturale, S/MIME è basato su:

- *PKCS-7* (S/MIME v2) e *CMS* (S/MIME v3): specifica le caratteristiche crittografiche ed i tipi di messaggi (equivalente al PEM).
- *PKCS-10* nel caso in cui si volesse usare la posta elettronica stessa per fare la richiesta di un certificato.
- *Certificati X.509* a supporto delle firme e della cifratura.

S/MIME - Algoritmi

Gli algoritmi usati sono:

- Digest: SHA-1 (obbligatorio) o MD5.
- Firma digitale: DSS (obbligatorio) o digest+RSA (soluzione più usata).
- Scambio chiavi: Diffie Hellman (obbligatorio). In realtà la pratica comune è quella di calcolare una chiave di messaggio cifrandola con la chiave RSA del destinatario.
- Cifratura del messaggio: 3DES o RC2/40 (entrambi obbligatori).

La preferenza per DSS e Diffie-Helmann, algoritmi in generale poco usati, deriva dal fatto che S/MIME è stato standardizzato dall'IETF, la quale desiderava specificare come obbligatori algoritmi che fossero liberi da brevetti, che non richiedessero il pagamento di royalties per implementarli.

MIME Type

Per supportare S/MIME sono stati definiti dei nuovi tipi MIME. In particolare:

- *application/pkcs7-mime*, usato per creare
  - Messaggi cifrati (il messaggio è codificato come un envelopedData PKCS-7).
  - Messaggi firmati binari (PCKS-7 signedData), destinati solo ad utenti S/MIME perché il messaggio è all'interno della busta PKCS-7 (non è in chiaro).
  - Messaggi che contengono solo una chiave pubblica, ovvero i cosiddetti PKCS-7 signedData "degenerati" che presentano la sezione dati nulla. Lo scopo è soltanto quello di trasmettere al destinatario il certificato.

  L'estensione standard da usarsi associata a questo tipo è .p7m, ed è sempre codificata in base64 perché altrimenti non potrebbe essere trasmesso su un normale canale SMTP.
- *multipart/signed*, usato per creare messaggi firmati destinati anche ad utenti non S/MIME. Con questa soluzione il messaggio resta in chiaro, e l'ultima parte MIME è la firma (come da RFC-18-47).
  La parte di firma ha estensione standard .p7s, ed è codificata in base64.
- *application/pkcs10*, che serve per inviare le richieste di certificazione ad una CA. anche questo tipo è codificato un base64.

S/MIME - Esempio di firma

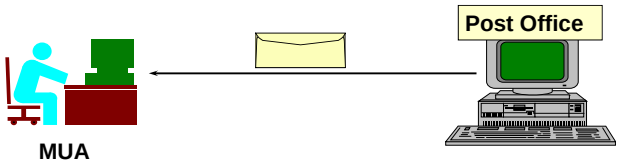
```
Content-Type: multipart/signed;
 protocol="application/pkcs7-signature";
 micalg=sha1;
 boundary="-----aaaaa"

-----aaaaa
Content-Type: text/plain
Content-Transfer-Encoding: 7bit

Ciao!
-----aaaaa
Content-Type: application/pkcs7-signature
Content-Transfer-Encoding: base64

MIIN2QasDDSdwe/625dBxgdhdsf76rHfrJe65a4f
fvVSW2Q1eD+SfDs543Sdwe6+25dBxfdER0eDsrs5
-----aaaaa-
```

Naming in S/MIME

Per poter utilizzare i certificati X.509 in supporto alla posta elettronica è indispensabile che all'interno del certificato ci sia l'indirizzo di posta elettronica. Altri identificati non sono utili, perché non associano la chiave con l'indirizzo di posta elettronica.

A tale scopo nell'S/MIME v2 veniva consigliato di inserire un campo *Email=* (oppure E=) come parte del DN. Era anche possibile usare opzionalmente l'estensione *subjectAltName* con codifica rfc822; con la v3 di S/MIME quest'ultima pratica è diventata obbligatoria.

9.8 Protocolli di accesso al MS



Nella modalità client-server il MUA ha la necessità di fare una query al suo Post Office (MS) per sapere se c'è della posta a lui destinata.

In uno schema del genere si hanno una serire di problemi:

- Autenticazione dell'utente da parte del server, in modo da evitare di dare accesso alla casella a qualcuno che non ne sia il legittimo proprietario.
- Autenticazione del server da parte del client, in modo che quest'ultimo possa essere certo di essersi collegato al suo vero Post Office.
- Riservatezza/integrità della posta elettronica, sia sul server sia durante il trasferimento in rete tra il Post Office e il MUA.

I protocolli usati per l'accesso al MS sono essenzialmente due:

- *Post-Office Protocolo (POP)*: ne esistono due versioni, POP-2 (non più utilizzato) e POP-3. L'autenticazione dell'utente viene fatta mediante l'utilizzo di una password in chiaro. Esiste una variante chiamata *APOP*, nella quale l'autenticazione dell'utente avviene mediante una sfida. Alcuni server supportano anche la variante KPOP, in cui si riesce a fare mutua autenticazione grazie ai ticket di Kerberos.
- *Internet Mail Access Protocolo (IMAP)*: l'autenticazione di default è data con username e password in chiaro. Opzionalmente è possibile usare OTP, Kerberos o la sua generalizzazione GSS-API.

9.8.1 Esempio POP-3

```
    telnet pop.polito.it 110
+OK POP3 server ready <7831.84549@pop.polito.it>
    USER lioy
+OK password required for lioy
    PASS antonio
+OK lioy mailbox locked and ready
    STAT
+OK 2 320
.........
    QUIT
+OK POP3 server signing off
```

1. Connessione al Post Office tramite la porta 110.
2. Il server risponde con un +OK (non si usano più i codici numerici come in SMTP) e si presenta all'utente.
3. L'utente inserisce la propria username preceduta dalla parola chiave *USER*.
4. Il server richiede all'utente di inserire la password.
5. L'utente inserisce la password preceduta dalla keyword *PASS*.

6. Il server fa la verifica dei dati inseriti dall'utente, e gli fornisce all'utente l'accesso alla propria casella postale.
7. L'utente utilizza il comando *STAT* per richiedere lo stato della casella postale.
8. Il server risponde dicendo che sono presenti due messaggi non ancora letti, per un totale di 320 byte.
9. L'utente esegue il comando *QUIT*.

È facile intuire che chi sia in grado di sniffare il canale di comunicazione vedrà passare in chiaro la username e la password dell'utente. POP-3 è quindi un protocollo altamente insicuro.

9.8.2 APOP

La variante APOP introduce un nuovo comando, chiamato appunto APOP, che sostituisce la coppia di comandi USER + PASS. Questo comando introduce un meccanismo a sfida di tipo simmetrico, dove la sfida è la parte della riga di hello racchiusa tra parentesi acute (incluse le parentesi). La sintassi del comando è la seguente:

APOP user risposta-alla-sfida

La risposta altro non è che un keyed digest, ossia *risposta = MD5(sfida + password)*. La risposta viene codificata in esadecimale, perché altrimenti non potrebbe essere trasmessa su un canale ASCII.

Il sistema più famoso che supportava APOP era Eudora.

Esempio APOP

```
   telnet pop.polito.it 110
+OK POP3 server ready <7831.84549@pop.polito.it>
   APOP lioy 36a0b36131b82474300846abd6a041ff
+OK lioy mailbox locked and ready
   STAT
+OK 2 320
.........
   QUIT
+OK POP3 server signing off
```

1. Connessione al Post Office tramite la porta 110.
2. Il server risponde con un +OK e si presenta all'utente.
3. L'utente decide di autenticarsi usando il comando APOP, e in particolare usa come sfida i caratteri <...> precedentemente inviati dal server.
4. Il server fa il medesimo calcolo dell'utente, e gli da accesso alla mail box.

Anche se APOP risolve il problema dello sniffing della username e della password, tutti gli altri dati rimangono comunque in chiaro.

9.8.3 IMAP

Per default il protocollo IMAP si basa su un processo di autenticazione molto debole:

LOGIN user password

Esiste però l'opzione di utilizzare uno di questi 3 comandi di seguito:

- *AUTHENTICATE KERBEROS_V4.*
- *AUTHENTICATE GSSAPI.*

- *AUTHENTICATE SKEY.*

L'unico caso in cui si ha mutua autenticazione è quando si usa Kerberos.

9.8.4 RFC-2595 (TLS For POP/IMAP)

Di tutte le opzioni possibili solitamente sia IMAP che POP utilizzano il metodo di autenticazione basato sulla trasmissione in chiaro dell'username e della password, e per tale ragioni sono soggetti a moltissimi attacchi.

Si è allora deciso di usare il protocollo TLS per proteggere il canale su cui poi verrà successivamente usato il protocollo POP o IMAP. In particolare l'RFC-2595 documenta su come usare TLS su questi canali.

Questo RFC ha aggiunto due nuovi comandi, STARTTLS per IMAP e STLS per POP3, da eseguire subito come primo comando in modo da trasformare il canale TCP normale in un canale protetto con TLS. Solo a questo punto verranno trasmessi username e password.

Con questa soluzione si ha il beneficio non soltanto di proteggere username e password, ma anche di proteggere tutti i trasferimenti della posta e di avere anche l'autenticazione del sever (obbligatoria in TLS).

9.8.5 Porte separate per SSL/TLS?

Quando si va a configurare un MUA per collegarsi ad un server è normalmente possibile scegliere tra 3 diverse opzioni:

- Usare un canale TCP, senza nessuna sicurezza.
- Usare STARTTLS, che permette di trasformare il canale creato in precedenza in un canale sicuro.
- Usare SSL, con il quale si va prima a creare il canale SSL su una porta dedicata e poi su questo verrà eseguito il protocollo applicativo.

La scelta di usare TLS o SSL influenza sul fatto che si avrà bisogno di porte separate. Infatti se si usa TLS sarà possibile usare la stessa porta, ma se si utilizza SSL si avrà bisogno di due porte separate, una per accedere al servizio normalmente e l'altra per accederci tramite SSL.

Porte separate per servizi sicuri/non sicuri viene sconsigliato normalmente dall'IETF per i seguenti motivi:

- Implicano URL diverse (http e https).
- Implicano un modello sicuro/insicuro non perfettamente corretto. Se ad esempio si utilizza https, il canale risultante potrebbe non essere sicuro perché potrebbe essere un canale a 40 bit; ciò implica che nonostante si usi https non si ha la certezza di avere un canale sicuro.
- Non è facile implementare l'idea di "usa SSL se disponibile", perché vorrebbe dire fare 2 tentativi di collegamento: se non si riesce a collegarsi alla porta SSL, si usa quella normale. Questo causa dei rellantamenti nell'attivare la connessione.
- Si va a raddoppiare il numero di porte necessarie.

Nella realtà avere porte diverse per servizi sicuri/insicuri presenta alcuni vantaggi:

- Semplicità di filtraggio sui firewall packet-filter, in quanto è sufficiente abilitare/disabilitare porte ben determinate.
- Eseguire SSL prima del protocollo applicativo permette di eseguire la client-authentication, la quale permette di non esporre le applicazioni ad attacchi.

Capitolo 10

Sicurezza delle applicazioni Web

Alcuni dei problemi riscontrabili nello sviluppo di applicazioni web sono:

- *SQL injection*, ossia l'esecuzione di query arbitrarie sul DB del server.
- *Cross-site scripting*, ossia l'accesso ai dati conservati nei cookie.
- *Accesso a dati riservati*, che tipicamente è un problema della gestione delle sessioni e più in generale di gestione nello stato del web.
- *Buffer overflow*, ossia l'esecuzione di codice arbitrario sul server (ed a volte anche sul client).

Tutti i problemi sopra citati sono tipicamente possibili perché nello sviluppare le applicazioni i developers tendono a fidarsi del client. In realtà questo è sbagliato, non bisognerebbe mai fidarsi del client perché controllare il client è assolutamente impossibile: il client è nelle mani dell'utente, che potrebbe potenzialmente essere un attaccante. Questo concetto è valido anche in una Intranet e anche per applicazioni protette da un firewall (il firewall protegge dove chiude le porte ma non dove le apre).

In generale quindi quando si sviluppa un'applicazione web accorrerebbe non fidarsi mai del codice che viene eseguito sul client, e soprattutto occorre sempre assucmere che i dati passati al client possano essere manipolati in modo inatteso.

In altre parole la sicurezza deve essere server-based, il client è tipicamente una fonte di insicurezza.

Inoltre visto che i dati vengono trasmessi usando una rete non sicura, durante la trasmissione in rete potrebbe capitare qualunque cosa. È quindi essenziale validare sempre i dati di provenienza non fidata, ossia validare i dati anonimi o dati che possono essere stati manipolati o falsificati.

Per la validazione dei dati è possibile seguire due tipi di approcci differenti:

- *Check that it looks good (whitelist)*, ossia andare a guardare se I dati corrispondono a quello che il server ha come concetto di dati validi.
- *Don't match that it looks bad (blacklist)*, ossia I dati non fanno match con quello che si sa essere un attacco.

La soluzione da preferire è quella della whitelist, perché andare a prevedere tutti i possibili modi con cui i dati possano essere alterati per realizzare un attacco è estremamente difficile. Dati non validati o non ripuliti prima di essere accettati dal server sono la sorgente di moltissimi attacchi.

10.1 SQL Injection

La SQL injection significa essenzialmente fornire un input artefatto per alterare il codice SQL generato dinamicamente da un server. In particolare l'attacco consiste in:

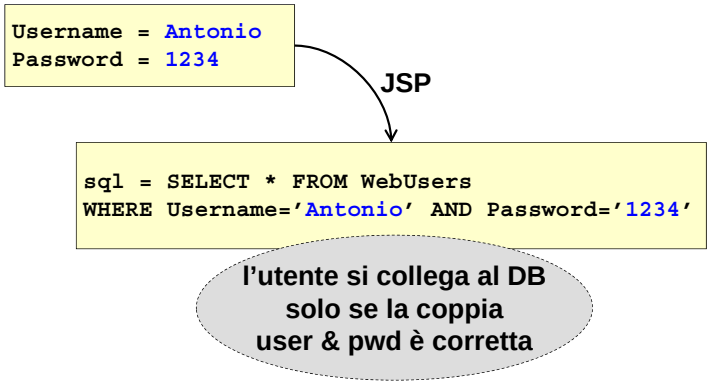
- Modificare le condizioni di una query.
- Selezionare dati fuori dalla tabella che teoricamente si stava usando.

Questo attacco viene usato per molti scopi, ma molto spesso per rubare le credenziali di autenticazione di un utente. Affinchè l'attacco vada a buon fine potrebbe essere necessario l'utilizzo di una qualche forma di social engineering.

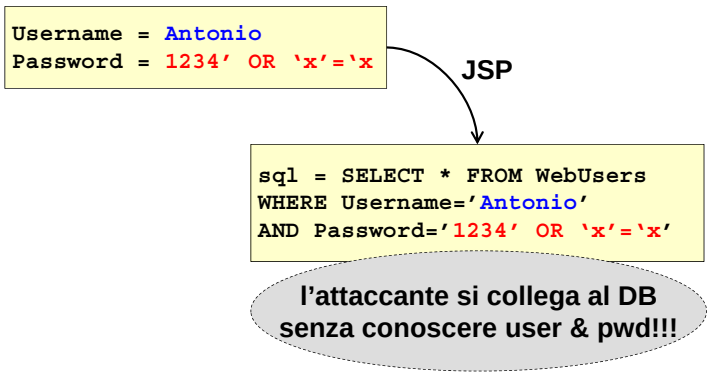
10.1.1 Esempio JSP n.1

```
String sql = new String(
  "SELECT * FROM WebUsers WHERE Username='"
  + request.getParameter("username")
  + "' AND Password='"
  + request.getParameter("password") + "'"
)
stmt = Conn.prepareStatement(sql)
rows = stmt.executeQuery()
le righe vengono inviate al browser ...
```

Nell'esempio si ha una query in cui si va a selezionare tutte le righe dalla tabella WebUsers, dove lo username corrisponde a quello introdotto nel campo "username" del form di autenticazione, e la password corrisponde a quella introdotta nel campo "password". Nelle due righe successive si prepara lo statement SQL e se ne fa l'esecuzione.



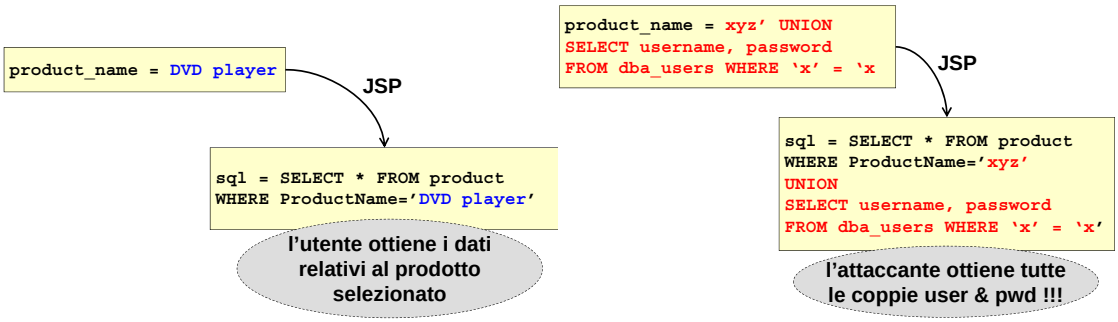
Nell'immagine sovrastante si ho un esempio di uso normale della procedura.



In questo caso invece un possibile attaccante esegue come password del codice opportuno, in modo che la query fornisca sempre un risultato positivo indipendentemente dal fatto che la password e lo username siano giusti.

10.1.2 Esempio JSP n.2

```
String sql = new String(
  "SELECT * FROM product WHERE ProductName='"
  + request.getParameter("product_name")
  + "'"
)
stmt = Conn.prepareStatement(sql)
rows = stmt.executeQuery()
le righe vengono inviate al browser ...
```



Grazie al fatto che non c'è nessun controllo sui nomi inseriti dall'utente, anche in questo caso l'attaccante riuscirà ad avere accesso ad informazioni riservate semplicemente andando ad inserire del codice ben studiato.

10.1.3 SQL Injection – Come evitarlo

Per cercare di evitare questo tipo di attacco esistono una serie di linee guida da seguire:

- Revisionare tutti gli script e tutte le pagine dinamiche, comunque generate (non solo quelle generate da JSP, ma anche quelle generate con PHP, ASP, ...).
- Validare l'input degli utenti, trasformando gli apici singoli in sequenze di due apici.
- Suggerire agli svilupaptori di usare le query parametrizzate (query SQL la cui struttura non può essere cambiata dai dati che vengono introdotti) per introdurre i valori delle variabili fornite dall'utente, piuttosto che generare codice SQL tramite concatenazione di stringhe.
- Usare software di testing per verificare la propria vulnerabilità a questo tipo di attacco.

10.1.4 SQL Injection – Dove può capire?

La SQL injection può capitare non solo nelle query dinamiche generate a partire da dei form web, ma anche in altri casi dove si va a creare delle query con dei parametri. Ad esempio nei Java Servlets, nelle Java Stored Procedures, in generale nei web services (SOAP, ...).

In generale quindi è possibile subire questo attacco in tutti quegli ambiti in cui si va a costruire un'azione a partire da dati ricevuti dall'utente.

10.2 Cross-site scripting

Anche noto come XSS (talvolta anche CSS), è un attacco usato per vari scopi, spesso per rubare le credenziali di autenticazione di un utente.

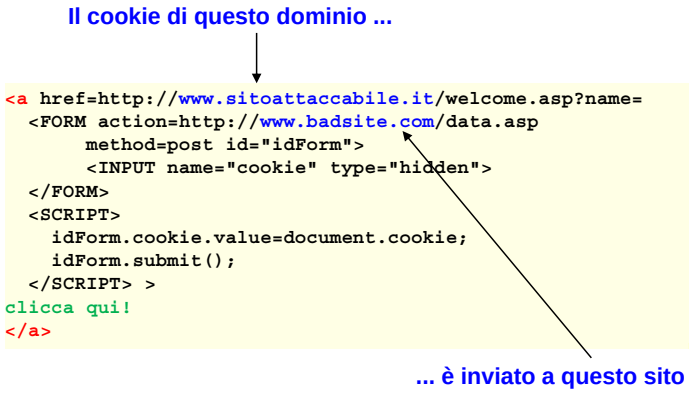
Affinchè l'attacco vada a buon fine è necessario usare una qualche forma di social engineering. È molto difficile da contrastare perché usa una grande varietà di meccanismi, ed è poco compreso dagli sviluppatori applicativi (anche data la complessità delle attuali applicazioni web). È un attacco molto più comune di quanto si pensi.

10.2.1 XSS – Come funziona

L'attaccante identifica un sito web che non filtra i tag HTML quando accetta input da un utente, così da andare ad inserire del codice HTML e/o degli script arbitrari in un link o in una pagina.

Uno degli esempi più clamorosi di questo attacco è quello che ha riguardato Ebay. L'attaccante andava a registrare un oggetto su Ebay ed andava ad incorporare lo script nella sua descrizione. Quando un utente andava a visualizzare la pagina relativa all'offerta, lo script veniva eseguito; lo script approfittava del fatto che l'utente era collegato con il cookie di Ebay per andare a fare un'offerta all'oggetto dell'inserzione senza che l'utente se ne accorgesse.

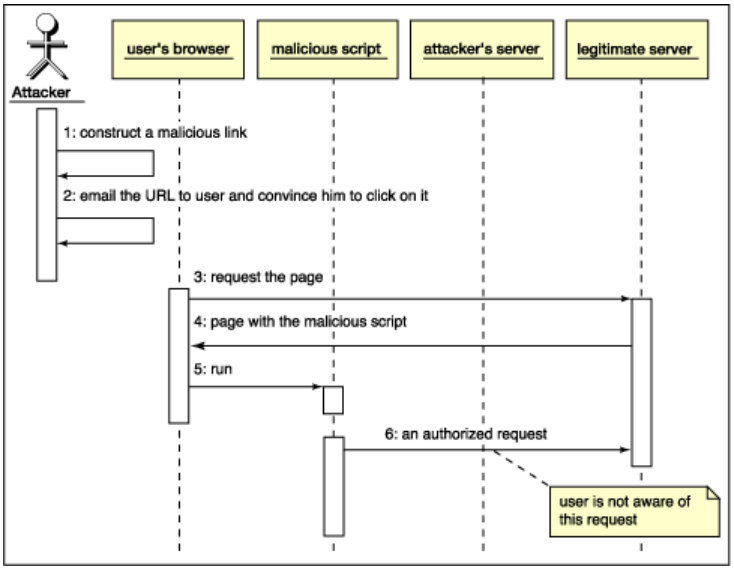
Questo attacco serve ad evitare la cosiddetta *Same-Origin Policy*, policy secondo la quale i browser permettono agli script del sito X di accedere solo ai cookie e ai dati provenienti dal sito X. Per tale ragione si fa eseguire alla vittima uno script per far inviare i suoi cookie (relativi al sito dello script) al sito web dell'attaccante.



Questo tipo di attacco è molto difficile da rilevare perché lo script può essere:

- Lasciato in chiaro.
- Offuscato (es. codifica esadecimale della stringa).
- Generato al volo ad esempio tramite un altro script.
- Nascosto dentro un messaggio di errore.

10.2.2 Sequence diagram



1. L'attaccante costruisce un link malizioso.
2. Fase di social enginnering, in cui l'attaccante invita la vittima a visitare il link precedentemente creato.
3. L'utente richiede la pagina.
4. Ottiene la pagina che contiene lo script malizioso.
5. Lo script viene eseguito.
6. Esecuzione di una qualche richiesta, di cui l'utente non è conscio.

10.2.3 XSS – Tipologie di attacco

Persistent XSS (stored XSS)

In questa versione dell'attacco l'attaccante usa un browser per inviare dei dati (es. tramite un form), che vengono memorizzati permanentemente sul server e poi essere inviati o resi disponibili ad altri utenti.

Il problema in questo caso è che i dati contengono tag HTML (soprattutto tag <script>) che modificano il contenuto della pagina.

Esempio di attacco è un post di un commento che contiene uno script per fare redirect verso un altro sito.

Nel 2008 una vulnerabilità di tipo persistent XSS su MySpace, unita con un worm, diffuso tramite un filmato QuickTime ha permesso l'esecuzione di codice JSP arbitrario usato per rubare le credenziali (phishing) ed il listing del contenuto del filesystem di una determinata persona.

Non-persistent XSS (reflected XSS)

L'attaccante invia dei dati tramite un browser, che vengono direttamente usati nell'applicazione lato server (es. in una query SQL) per generare la risposta. Si ha quindi il problema di generare dell'input inatteso all'applicazione, che va ad eseguire un'operazione imprevista.

SQL injection è un esempio di non-persistent XSS, in quanto questo attacco non va a modificare permanentemente il server, ma va a modificare il formato dell'operazione della query SQL.

DOM XSS

L'attaccante invia dei dati tramite un browser, che vengono memorizzati o usati direttamente dal server per generare la risposta.

Il problema è che i dati contengono dei tag che manipolano il DOM (Document Object Model), ad esempio usando una XmlHttpRequest, che provocano un compartamento inatteso lato client.

10.2.4 XSS – Contromisure

In generale se si vuole evitare di subire degli attacchi XSS occorre:

- Sempre validare l'input, andando ad accettare solo l'input corretto e rifiutare (MAI cercare di correggere) input errato.
- Codificare l'output con le entità HTML; ad esempio se si vuole inserire l'apice si utilizzi "&quot", che graficamente rappresenta l'apice ma non è interpretabile come un tag SQL. Non bisogna quindi mai usare codici ASCII o UTF-8.
- Specificare sempre la codifica dell'output per evitare interpretazioni errate.
- Non usare mai le blacklist ma sempre le whitelist, principalmente perché p difficile scrivere correttamente una blacklist che tenga conto di tutti i possibili tipi di attacco.
- Canonicalizzare l'input prima di esaminarlo, questo per evitare errori di interpretazione.

10.2.5 XSS – Alcuni controlli base

Innanzitutto esistono librerire anti-XSS, specifiche dei vari linguaggi di programmazione. Ad esempio sia in .NET che in PHP quando si riceve un input è possibile chiamare una funzione di un'apposita libreria che controlla se l'input nasconda qualcosa di malevolo. Questo tipo di controllo è per i "pigri", in quanto si basa su una blacklist: vengono ricercati se i dati ricevuti contengono delle informazioni associabili ad un attacco XSS.

L'idea migliore sarebbe quello di creare una whitelist (tipicamente un'espressione regolare) per ogni valore di input che si riceve.

Un'altra soluzione è quella di ripulire l'input ricevuto, andando a filtrare i metacaratteri, e andare a convertire qualunque testo salvato o letto.

Da ...	a ...
<	<
>	>
#	#
&	&
(	(
)	)

Figura 10.2: Se l'utente ha inserito un carattere che potrebbe essere pericoloso, è buona norma andare a sostituirlo con un'entità HTML che risulta non essere pericolosa.

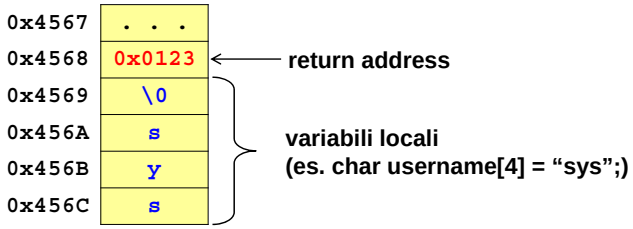
10.3 Buffer overflow

In questo tipo di attacco si cerca di fornire più dati del massimo atteso per superare i limiti della memoria allocata e far eseguire codice arbitrario.

Buffer overflow, integer overflow o errori nelle stringhe di format sono molto importanti per programmi scritti in C/C++, quali ad esempio le applicazioni custom o I server web stessi.

Per capire come funziona il buffer overflow bisogna ricordarsi il funzionamento dello stack. Ad ogni chiamata di procedura/funzione:

- Si salva nello stack l'indirizzo di ritorno;
- Si alloca sullo stack le variabili necessarie alla funzione.



10.3.1 Buffer overflow – Esempio

```
void BO_example (char *prod_name)
{
  char query[100] =
    "SELECT * FROM product WHERE ProductName='";
  strcat (query, prod_name);
  strcat (query, "'");
  // now exec query
  ...
}
```

Se prod_name contiene più di 100-43=57 byte si ha un buffer overflow:
- la query viene eseguita correttamente ...
- ... ma al termine della funzione si esegue il codice macchina contenuto dalla posizione 58 di prod_name

Questo è la tipologia di attacco che si è avuta nei confronti dell'IS, in cui se si aveva una stringa nella URL più lunga di 256 caratteri si andavano a sovrascrivere i dati presenti in memoria ed era possibile eseguire del codice arbitrario.

10.3.2 Buffer overflow – Contromisure

Le contromisure principali per il buffer overflow sono:

- Buona programmazione, andando a non usare ad esempio funzioni come strcpy, strcat, ... ma strncpy, strncat, ...
- Usare sistemi automatici di protezione
  - Modificare il compilatore per proteggere l'indirizzo di ritorno mettendogli attorno dei "canarini", ossia due stringhe che vengono messe subito prima e subito dopo l'indirizzo di ritorno. Se quando la procedura finisce i canarini non sono più vivi (non hanno più lo stesso valore iniziale), è un'indicazione che sono stati sovrascritti e occorrerà bloccare l'esecuzione del programma.
  - Configurare lo stack come "no exec" (default il Solaris 9).

  Esistono tuttavia dei nuovi metodi di attacco che sono in grado di aggirare queste protezioni, adando ad esempio a cambiare l'indirizzo di ritorno con l'indirizzo che attiva una DLL. La soluzione migliore quindi è quella di scrivere bene il codice.
- Esaminare i log per cercare tentativi di attacco.

10.4 OWASP top 10 web risks (2013)

OWASP è un progetto completamente OpenSource che mira a sviluppare tecniche, soluzioni ma anche solo conoscenza circa la sicurezza delle applicazioni web. Tra le tante iniziative di questo progetto si ha la testura periodica di una lista dei 10 attacchi più frequenti.

A1. Injection
A2. Broken authentication and session management
A3. Cross-Site Scripting (XSS)
A4. Insecure direct object references
A5. Security misconfiguration
A6. Sensitive data exposure
A7. Missing function-level access control
A8. Cross-Site Request Forgery (CSRF)
A9. Using components with known vulnerabilities
A10. Unvalidated redirects and forwards

10.4.1 Injection

SQL injection è sicuramente il problema più famoso, ma sono possibili anche altri tipi di attacco in quanto in generale le applicazioni web usano la costruzione di stringhe per molte cose. È quindi possibile eseguire attacchi tramite LDAP (usa una stringa per fare una query all'interno del directory), XPath (per selezionare parti di un DOM), XSLT, HTML, CML, OS command, ...

Problema: avere dell'input inatteso eseguito da un interprete.
Soluzioni:

- Evitare (per quanto possibile) gli interpreti.
- Usare comandi a struttura fissa, usando l'input dell'utente solo come un parametro.

10.4.2 Broken authentication and session management

Al secondo posto si hanno problemi riguardanti l'autenticazione e la gestione delle sessioni, principalmente dovuti al fatto che si utilizzano schemi non standard di autenticazione che contengono spesso degli errori. Progettare degli schemi di autenticazione in maniera corretta è molto difficile. Le aree più comuni dove è possibile avere degli errori sono il log-out, la gestione delle password, timeout, la funzione "ricordami su questo computer" e l'aggiornamento dell'account.

Tutti questi problemi causano l'esposizione delle password o degli account e degli identificativi di sessione, permettendo di ottenere i privilegi della vittima e spesso di controllare molte altre sessioni.

Sono in generale attacchi difficili da individuare perché utilizzano molte tecniche diverse.

Questi attacchi potrebbero in generale essere contrastati andando ad usare degli schemi di autenticazione standard, cercando di avere cura dei cookie di sessione andando ad eliminarli non appena non sono più utilizzati.

10.4.3 Insecure direct object references

Può capitare che le applicazioni usino direttamente come parametro un oggetto reale (es. una chiave di DB, un nome di un file), andando ad esporre questa informazione all'utente. Il client potrebbe andare a cambiare tale riferimento per cercare di accedere ad un diverso oggetto.

Ad esempio nel 2000 il server web dell'ufficio Australiano delle tasse usava nella URL il codice fiscale dell'utente. Sfruttando questa anomalia un utente, autenticato con le sue credenziali, è riuscito a raccogliere i dati fiscali di altri 17000 utenti. Questo attacco è stato possibile perché

non c'era nessuna correlazione tra i dati di autenticazione e il codice fiscale, non aveva nessun senso usare come parametro il codice fiscale.

Soluzione: Non esporre mai nell'interfaccia accessibile all'utente i riferimenti agli oggetti applicativi

Esempio

Query SQL composta con input fornito dall'utente.

```
String query =
  "SELECT * FROM accts WHERE account = ?";
PreparedStatement pstmt =
     connection.prepareStatement(query , ... );
pstmt.setString (1, request.getParameter("acct"));
ResultSet results = pstmt.executeQuery( );
```

Cambiando il parametro "acct" dal proprio browser l'attaccante può accedere a qualsiasi altro account.

Security misconfiguration

Se un sistema non è stato configurato a dovere, un attaccante potrebbe essere in grado di accedere al sistema o di raccogliere informazioni riservate. Tipici problemi di malconfigurazione sono:

- Uso di credenziali di default.
- Pagine non più usate ma che non sono state rimosse dal server.
- Nessuna precauzione contro vulnerabilità note.
- File e cartelle non protetti; solo per il fatto che qualcosa non è esposto non significa che non sia accessibile.

Errori di configurazione possono presentarsi a qualunque livello dello stack applicativo. Questo significa che gli sviluppatori devono avere una stretta collaborazione con gli amministratori di sistema, in modo da assicurarsi che l'intero stack sia correttamente configurato, andando anche ad usare degli scanner automatici.

Scenari

Possibili scenari di attacco possono essere:

- Utilizzo di framework vulnerabili ad attacchi XSS, magari a causa dell'uso di librerie non aggiornate.
- Account di amministrazione creato automaticamente, usando username e password standard.
- Directory listing non disabilitato, che potrebbe essere sfruttato da un attaccante per scaricare tutto il codice sorgente ed identificare le vulnerabilità più facilmente.
- Visualizzazione da parte del server web dello stack trace in caso del verificarsi di qualche errore, che potrebbe essere sfruttatto da un attaccante per ricavare informazioni utili.

10.4.4 Sensitive data exposure

I dati sensibili vanno protetti in modo particolare sia quando memorizzati permanentemente sia in trasmissione. Si parla in particolare di data at rest, ossia memorizzati sull'unità di memoria, e di data in transit, quando i dati vengono trasmessi.

Tipici problemi per i data at rest:

- Dati sensibili non cifrati o cifrati con algoritmi "casalinghi".
- Uso errato di algoritmi forti.
- Uso di algoritmi notoriamente deboli, come RC2-40.
- Chiavi memorizzate direttamente nell'applicazione o in file non protetti.
- Memorizzare dati non necessari. Bisognerebbe cercare si seguire sempre il principio del need to know, ossia memorizzare solo i dati che servono veramente per l'applicazione che si sta eseguendo.

Per quanto riguarda la parte di data in transit, è necessario andare a considera l'uso solo di canali protetti, sia per le sessioni autenticate verso i client, sia per i collegamenti verso il back-end.

Spesso non viene fatto l'uso di un canale sicuro per un presunto peggioramento delle prestazioni (in parte questo è vero), o per conflitti con gli IDS/IPS. Spesso SSL/TLS è usato solo durante la fase di autenticazione, o con certificati scaduti o non configurati correttamente.

Scenari (Data at rest)

- Cifratura dei dati delle carte di credito quando si va a memorizzarli all'interno di un DB, andando però ad utilizzare delle query li decifrano. Un attaccante può quindi ottenere i dati in chiaro tramite un attacco SQL injection. La soluzione più ovvia è quella di non usare mai query che decifrano i dati.
- Backup su CD di alcuni dati sanitari, andando però ad inserire la chiave sullo stesso backup. Per decifrare i dati è quindi sufficiente entrare in possesso del supporto ottico.
- Creare il DB delle password senza usare il salt. Ad esempio decifrare il file /etc/passwd con un attacco brute force richiede 4 settimane, invece che 3000 anni!

Scenari (Data in transit)

- Siti che non proteggono bene la pagina che richiede l'autenticazione, rendendo la vita molto semplice all'accante che dovrà semplicemente intercettare il traffico di rete, andando ad intercettare il cookie di sessione per impersonare la vittima.
- Sito che utilizza un certificato scaduto. L'utente è abituato all'avvertimento "Attenzione il certificato è scaduto, vuoi procedere comunque?", e non si accorge nel momento in cui viene rediretto su un sito diverso e gli comunica le sue credenziali (phishing).
- Connessione a DB tramite oggetti ODBC/JDBC è molto perisoloso perché tutte le comunicazioni avvengono in chiaro.

10.4.5 Missing function-level access control

Questo problema riguarda la mancanza di controlli di accesso a livello della singola funzione. In particolare la protezione di una URL, o di un'azione richiesta tramite una URL (es. ?action=payment), viene fatta semplicemente non rendendola pubblica o non presentandola nell'interfaccia (implementazione del concetto security through obscurity), non andando a fare nessun controllo reale di accesso o andando a fare un semplice controllo client-side o un controllo basato solo su i dati forniti dal client. Questo rende i dati facilmente accessibili a tutti.

Problema: protezione facilmente superabile.

10.4.6 Scenari

Si supponga di avere le pagine

http://example.com/app/getappInfo
http://example.com/app/admin_getappInfo

La prima fornisce informazioni di base, mentre la seconda dovrebbe essere riservata agli amministratori di sistema.

getappInfo è disponibile a tutti, ma il link che porta a admin_getappInfo richiama una procedura di autenticazione. Tuttavia la procedura di autenticazione non viene attivata se si scrive direttamente la URL nel browser,e quindi è sufficiente indovinare il nome del file per superare l'accesso.

10.4.7 Cross-Site Request Forgery (CSRF)

Questo attacco consiste nell'inviare al browser una URL attiva (contenuta ad esempio in un sito web cattivo o in una mail HTML) per eseguire un'azione sul server bersaglio.

```
<img src="http://miabanca.it/do?azione=bonifico">
<img src="http://example.com/app/transferFunds?
amount=1500&destinationAccount=attackerAcct#"
width="0" height="0" />
```

L'attacco è estremamente efficace verso quei server web che usano autenticazione automatica, ad esempio basata su cookie dopo login, su ticket di Kerberos o su l'invio automatico di username e password.

10.4.8 Using components with known vulnerabilities

Oggi le applicazioni web sono molto complesse e usano molti componenti, come librerie per trattare i dati lato server o lato client, framework, servizi, ecc.

Le vulnerabilità di queste componenti sono spesso note e risolte ma raramente gli sviluppatori aggiornano le applicazioni (re-build), anche perché spesso non sanno quali librerie stanno usando a causa di dipendenze profonde e nascoste. Questo permette l'uso di strumenti di attacco automatici, ed estende la platea degli attaccanti anche a persone senza le necessarie conoscenze.

Occorre quindi implementare un adeguato Component Lifecycle Management (CLM): bisogna sapere tutti i componenti che si stanno usando, tenere traccia delle loro vulnerabilità, e ogni volta che esce fuori un aggiornamento occorre fare il re-build di tutte le applicazioni che usavano quel determinato componente.

10.4.9 Unvalidated redirects and forwards

Questo attacco consiste nel forzare la vittima ad usare un link con un redirect non validato. Se il link appartiene a un sito valido la vittima si fida. Ma in realtà la pagina bersaglio è specificata in un parametro non validato, e permette quindi di scegliere arbitrariamente la pagina di destinazione.

Questo attacco è molto facile da rilevare se si ha un controllo su i redirect, ma non è così semplice se si usano i forward (uso interno allo stesso sito).

Scenari

Si supponga di avere un sito con una funzione "redirect.jsp", che riceve un singolo parametro "url" contenente l'indirizzo della prossima pagina da eseguire. L'attaccante crea un URL malevolo per installare malware o per fare phishing

http://www.x.com/redirect.jsp?url=evil.com

Se la funzione redirect.jsp non contiene nessun controllo, la vittima che riceverà l'URL malevolo e la eseguirà verrà rediretto verso una destinazione non validata.

Un sito potrebbe anche usare la funzione di forward per girare richieste tra diverse parti del sito tramite un parametro (es. se una transazione è avvenuta correttamente o meno).

L'attaccante sfrutta questo a suo vantaggio andando a creare una URL per accedere ad una pagina alla quale non potrebbe accedere direttamente

http://www.x.com/boring.jsp?fwd=admin.jsp

Nell'esempio l'attaccante fa forward sulla pagina di admin. Normalmente non avrebbe le credenziali per accedervi, ma se il motore non fa nessun controllo tra la sua identità e le pagine a cui può avere accesso riuscirà ugualmente ad accedervi.

Capitolo 11

Anonimato in Internet

L'anonimato in internet è un problema particolarmente caldo in questo periodo perché è una moneta con due facce: da una parte l'anonimato è una cosa importante per proteggere la libertà di espressione e la libertà dei cittadini in tutti quanti quei paesi dove c'è una forte censura o dove ci sono dei regimi autoritari che vogliono impedire l'espressione di idee diverse da quelle del governo, dalla'ltra parte però le tecniche di anonimato possono anche essere sfruttate da persone malintenzionate per nascondere le loro azioni e quindi rendere difficili tutte le attività di investigazione.

Anonimato in Internet significa non essere identificabile all'interno di un certo insieme; non si può essere anonimi da soli, l'anonimato è sempre all'interno di un certo gruppo di utenti.

L'anonimato si distingue in due forme:

- *Forward anonymity*: impossibile conoscere il server contattato da un certo utente noto.
- *Reverse anonimity*: impossibile conoscere gli utenti che contattano un server noto.

Per realizzare l'anonimato esistono varie tecniche:

- *End-to-end encryption*, tramite ad esempio l'uso di IPsec. Questa soluzione non è applicabile a servizi pubblici, ma solo ad utenti che hanno fatto degli accordi preventivi.
- *Anonymizing proxy*, ossia sfruttare dei proxy che implementano l'anonimato a cui un utente si collega ed esegue le operazioni per suo conto. Questa soluzione è applicabile assumendo che il traffico tra l'utente e il proxy non venga monitorato, ma introduce due problemi: il proxy diventa un single point of failure, ed inoltre è importante sapere il livello di fiducia del gestore del proxy.
- *VPN*, nel caso in cui si è a conoscenza che solamente una porzione della rete è monitorata o non fidata.
- *Darknet*, le cosiddette reti oscure in cui il traffico è completamente non osservabile seguendo varie tecniche. Il sistema più noto è TOR, ma esistono anche I2P e Freenet.

11.1 The Onion Routing (TOR)

TOR è un progetto sviluppato inizialmente dalla US Navy tramite due centri di ricerca (CHACS e NRL). Lo scopo iniziale era quello di rendere inosservabili e anonime le sue comunicazioni.

Il progetto è stato poi rilasciato open-source, ed è stato immediatamente adottato dalla EFF. Questo ne ha garantito una rapida diffusione, ed oggi è usato da tantissimi tipi di utenti:

- Giornalisti, per proteggere le loro fonti.
- Governi, per fare open-source intelligence (monitoraggio delle reti in anonimato).

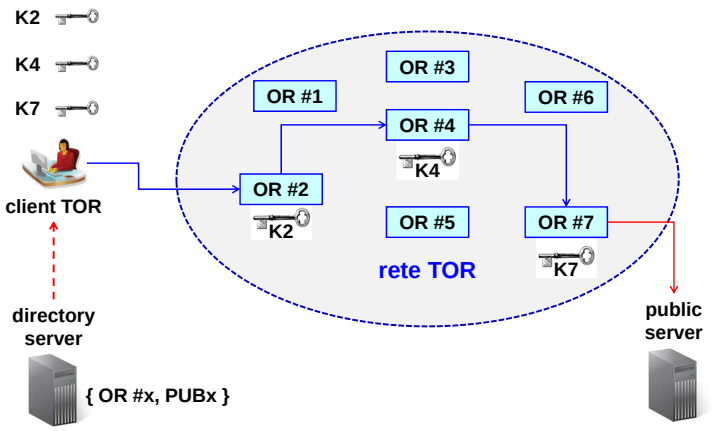
- Attivisti, per proteggersi dalla censura.
- Militari, per fare comunicazioni su reti pubbliche in modo anonimo.

11.1.1 TOR – Elementi base

Gli elementi essenziali di TOR sono:

- Una overlay network per nascondere mittente e destinatario, tramite una serie di nodi intermedi detti Onion Router (OR).
- Ogni OR è collegato al prossimo OR del circuito virtuale tramite TLS a fini di sicurezza (anche se in realtà è a fini di integrità e di autenticazione, in modo che non ci siano dei nodi che non appartengono alla rete che si intromettano).
- Gli utenti usano degli Onion Proxy (OP) per collegarsi agli OR.
- Gli OP creano dei circuiti virtuali (VOP, Virtual Onion PAth), che attraversano un certo numero di OR scelto dall'utente. Tanti più nodi intermedi l'utente andrà ad attraversare, tanto più crescerà il suo livello di anonimato; bisogna trovare un compromesso tra latenza e livello di anonimato desiderato.
  - La comunicazione tra un OR e il successivo avviene tramite dei segmenti di dati, cifrati con AES + chaivi effimere DH.
  - VOP vengono cambiati periodicamente (tipicamente ogni 600 s), anche tra gli stessi OP e server.

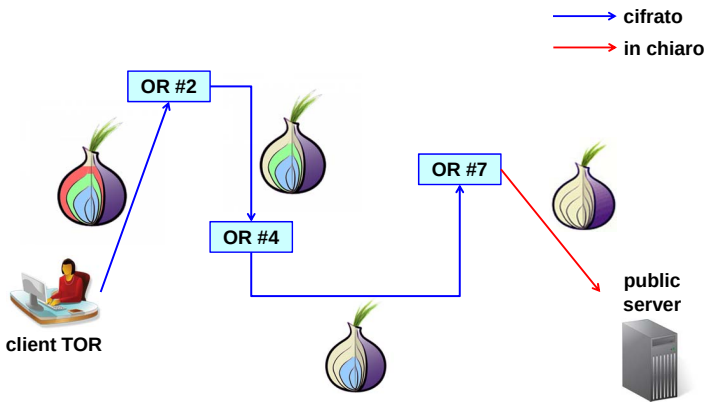
11.1.2 TOR – Circuito virtuale



Innanzitutto il client TOR usa un directory server per conoscere quali sono le chiavi pubbliche di tutti gli OR. Dopodichè andrà a selezionare un certo numero di OR da attraversare; nell'esempio il client ha scelto gli OR #2, #4 e #7. Gli OR scelti avranno delle loro chiavi pubbliche, e di conseguenza il client andrà a creare 3 chiavi dinamicamente (K2, K4, e K7) per cifrare i messaggi destinati ai vari OR.

Alla fine della rete TOR ci sarà un OR, il punto di uscita, che colloquierà con il server pubblico, non facendo rimanere nessuna traccia di quale sia l'IP da cui la comunicazione è iniziata.

11.1.3 TOR – Onion routing



Il sistema viene chiamato onion routing perché l'utente va a creare una cifratura con una struttura simile ai vari strati di una cipolla. Gli strati verrano tolti ad uno ad uno man mano che si procede.

Guardando l'esempio, lo strato più esterno (rosso) era la cifratura che l'utente aveva concordato con l'OR #2. Nel momento in cui questo riceve il messaggio, toglie lo strato più esterno e l'unica cosa che vede è che il passo successivo nella comunicazione è l'OR #4; i dati continuano ad essere cifrati. In altre parole l'OR #2 potrà solamente mandare il pacchetto al prossimo OR della catena. Ricevuto il pacchetto, l'OR #4 ne rimuove lo strato più esterno e scopre che il prossimo OR da contattare è il #7. Quest'ultimo ricevuto il pacchetto ne rimuove l'ultimo strato di cifratura e lo andrà ad inoltrare in chiaro al server pubblico.

Con questa struttura si ha che il primo nodo della catena conosce l'indirizzo del client ma non quello del server, mentre l'ultimo nodo conosce il server ma non il client. I nodi intermedi non hanno invece nessuna informazione sul client o sul server.

11.1.4 Hidden service

Lo schema implementato da TOR permette ad un client di accedere in modo anonimo ad un server pubblico. Tuttavia esistono alcuni server che non hanno il piacere di essere pubblici, ad esempio per evitare attacchi di tipo DDoS o anche attacchi fisici alla postazione presso cui sono ubicati.

In una situazione del genere TOR fornisce dei servizi, gli hidden service, che servono a nascondere l'identità logica e fisica dei server. Gli utenti quindi non sanno dove sia il sito né chi lo gestisce, e allo stesso tempo il gestore del sito non sa chi sono gli utenti del suo sito.

Questo sistema è implementato in TOR con gli Introduction Point (IP) e i Rendez-vous Point (RP).

Hidden service – Implementazione

1. Bob gestisce un servizio nascosto.
2. Bob sceglie una serie di IP (che altro non sono che degli OR) e crea dei VOP dal suo server verso di essi.
3. Bob rende noti ad Alice (o a tutti) gli IP.
4. Alice sceglie un RP e crea un VOP verso di esso.
5. Alice si collega ad un IP e comunica quale sia il suo RP a Bob.
6. Se Bob accetta di comunicare con Alice, crea un VOP verso il RP.
7. Il RP si chiama in questo modo perché connette i due canali di Alice e Bob.

Gli IP quindi non servono a svillupare davvero il traffico, ma servono soltanto per ricevere richieste di collegamento contenti l'indicazione di quali siano i RP. Sarà il server a decidere se accettare o meno una richiesta di collegamento.

11.1.5 The TOR browser

Dal punto di vista dell'utente il prodotto TOR più importante è il TOR browser, ache noto come Tor Browser Bundle (TBB).

Si tratta di una versione di Mozilla Firefox, modificato con alcuni bottoni speciali (TorButton e TorLauncher), contenente soprattutto due estensioni standard molto importanti anche per utenti non-TOR:

- *"Noscript" extension.*
- *"HTTPS everywhere" extension.*

Senza queste due estensioni il browser è vulnerabile a vari attacchi basati su script. Ad esempio l'FBI nella sua lotta contro Eric Eoin Marques (più grande distributore di materiale pedopornografico al mondo) ha scoperto che lui usava TOR per tenere nascosto le proprie comunicazioni, ma non aveva attivato euste due estensioni. L'FBI è allora riuscita a creare delle pagine fasulle con degli script javascript (attacchi simili al XSS concettualmente), i quali quando venivano eseguiti sulla macchina del ricercato inviavano quali fossero il suo MAC, il suo IP, il suo hostname, ecc. Questo ha permesso all'FBI di rintracciarlo e arrestarlo.

"Noscript" extension

Estensione scritta e mantenuta da un italiano, che fa parte del team di sviluppo di Firefox, che essenzialmente esegue i contenuti attivi (JS, Java, Flash, ...) solo se questi sono contenuti all'interno di siti inseriti all'interno di una whitelist.

Lo script serve inoltre per proteggersi da altri attacchi al browser. Ad esempio ha di default una serie di protezioni contro XSS, CSRF, clickjacking, ...

Fornisce anche altri vari benefici, ma in generale per quanto riguarda TOR quello più importante è il primo elencato.

"HTTPS everywhere" extension

Script sviluppato dall'EFF che cambia tutte le URL "http" in "https" se il servizio è disponibile tramite SSL.

In particolare òa EFF mantiene un sito che lista tutti i siti web del mondo che sono disponibili sia con http sia con https. In questo modo consultando questa lista il browser può scegliere se eseguire o meno il servizio richiesto tramite SSL.

Oltre a questo, visto che SSL si basa moltissimo sulle CA, questa estensione fa anche un controllo supplementare sulle CA che sono state compromesse e che invece l'utente potrebbe ancora avere valide nel suo browser.

11.2 Freenet

Freenet è una rete P2P progettata per garantire libertà di parola, combattendo quindi la censura di Internet. Gli utenti sono autenticati ma usano degli pseudonimi, e per evitare che qualcuno possa cancellare del materiale i dati sono replicati su tantissimi nodi. Il concetto è quindi quello di volere far una pubblicazione in modo anonimo per non essere perseguibile dal governo, e si vuole che tale publicazione non venga cancellata.

Freenet ha due modalità operative, che possono anche essere usate simultaneamente:

- *Opennet*, in cui un nodo si può collegare ad un qualunque altro nodo.

- *Darknet*, in cui si utilizzano collegamenti cifrati solo tra un sottoinsieme di nodi che sono stati dichiarati come fidati.

I documenti sono firmati dall'autore (in modo che nessuno lo possa modificare) tramite il suo pseudonimo, divisi in chunk i quali vengono memorizzati cifrati su vari nodi. Questo significa che l'operatore di in nodo non sa nemmeno quali dati sta ospitando, e quindi anche per la legge non è responsabile.

Proprio perché i dati sono replicati su più nodi è impossibile cancellarli. Per questa ragione quando si vuole cancellare una vecchia versione si va a caricarne una nuova. L'unico modo in cui i dati possono essere cancellati è se viene a mancare spazio sui dischi dei nodi Freenet. Questi hanno la possibilità di "dimenticarsi" dei dati più vecchi che non vengono più richiesti da tanto tempo.

Freenet viene in generale usato per creare dei blog, ma anche dei siti interi. Questi siti sono accessibili mediante dei browser modificati, che hanno lo scopo di cercare dove sono stati memorizzati i dati all'interno della rete Freeenet.

Appendice A

Definizioni

A.1 Introduzione alla sicurezza delle reti e dei sistemi informativi

sicurezza informatica ✡ l'insieme dei prodotti, dei servizi, delle regole organizzative e dei comportamenti individuali che proteggono i sistemi informatici di un'azienda

disponibilità ✪ la proprietà di essere accessibile e utilizzabile su richiesta di un'entità autorizzata

tracciabilità ✯ la proprietà di un sistema o risorsa di sistema che garantisce che le azioni di un'entità di sistema possono essere tracciate univocamente a quell'entità, che può quindi essere ritenuta responsabile delle sue azioni

autenticazione ☆ il processo di verifica di una dichiarazione che un'entità di sistema o una risorsa di sistema ha un certo valore di attributo

autenticazione dell'entità pari ✪ la corroborazione che un'entità pari in un'associazione è quella dichiarata

autenticazione dell'origine dati ✪ la corroborazione che la sorgente dei dati ricevuti è come dichiarato

ripudio ✪ negazione da parte di una delle entità coinvolte in una comunicazione di aver partecipato in tutta la comunicazione o in parte di essa

autorizzazione ☆ un'approvazione che è concessa a un'entità di sistema per accedere a una risorsa di sistema

controllo accessi ☆ protezione di risorse di sistema da accessi non autorizzati

riservatezza ✪ il diritto degli individui di controllare o di influenzare quali informazioni relative ad essi possono essere raccolte e memorizzate e da chi e a chi possono essere divulgate quelle informazioni

confidenzialità ✪ la proprietà che le informazioni non sono rese disponibili né divulgate a individui, entità o processi non autorizzati

integrità dei dati ✪ la proprietà che i dati non sono stati alterati o distrutti in una maniera non autorizzata

replay attack ☆ un attacco in cui una trasmissione dati valida viene ripetuta in modo malevolo o fraudolento, o dall'originatore o da un terzo che intercetta i dati e li ritrasmette, eventualmente come parte di un attacco masquerade

servizio di sicurezza ☆ un servizio di elaborazione o di comunicazione che è fornito da un sistema per dare uno specifico tipo di protezione alle risorse di sistema

asset ✡ l'insieme di beni, dati e persone necessarie all'erogazione di un servizio informatico

vulnerabilità ✡ debolezza di un asset

minaccia ✡ evento intenzionale o accidentale che può causare la perdita di una proprietà di sicurezza sfruttando una vulnerabilità

evento (negativo) ✡ verificarsi di una minaccia di tipo "evento accidentale"

attacco ✡ verificarsi di una minaccia di tipo "evento intenzionale"

incidente ✯ un evento di sicurezza che compromette l'integrità, la confidenzialità o la disponibilità di un asset informativo

breccia ✯ un incidente che comporta la diffusione o la potenziale esposizione di dati

diffusione di dati ✯ una breccia per la quale è stato confermato che i dati sono stati realmente diffusi (non solo esposti) a un party non autorizzato

hacker ✡ una persona che si diverte ad esplorare i dettagli dei sistemi programmabili e come allungarne le capacità, a differenza della maggioranza degli utenti, che preferiscono imparare solo lo stretto necessario

cracker ✡ una persona che sfrutta la propria conoscenza informatica per rompere la sicurezza in un sistema

masquerade ☆ un tipo di azione di minaccia per mezzo del quale un'entità non autorizzata ottiene l'accesso a un sistema o esegue un atto malevolo fingendosi illegittimamente un'entità autorizzata

spoof ☆ tentativo di un'entità non autorizzata di ottenere l'accesso a un sistema fingendosi un utente autorizzato

wiretapping ☆ un attacco che intercetta le informazioni contenute in un flusso di dati e accede ad esse in un sistema di comunicazione

wiretapping passivo ☆ tenta solo di osservare il flusso di dati e ottenere la conoscenza delle informazioni contenute in esso

denial of service ☆ l'impedimento dell'accesso autorizzato a una risorsa di sistema o il ritardamento delle operazioni e delle funzioni del sistema

calcolo distribuito ☆ una tecnica che disperde un singolo insieme di compiti logicamente correlati tra un gruppo di computer geograficamente separati ma cooperanti

zombie ☆ un computer host Internet che è stato penetrato surrettiziamente da un intruso che ha installato software demone malevolo per far sì che l'host operi come un complice nell'attaccare altri host, specialmente negli attacchi distribuiti che tentano il denial of service tramite l'inondazione

shadow server ✡ host che riesce a mostrarsi alle vittime come fornitore di un servizio senza averne il diritto

wiretapping attivo ☆ tenta di alterare i dati o di influenzare altrimenti il flusso

hijack attack ☆ una forma di wiretapping attivo in cui l'attaccante assume il controllo di un'associazione comunicativa precedentemente stabilita

attacco man-in-the-middle ☆ una forma di attacco di wiretapping attivo in cui l'attaccante intercetta e modifica selettivamente i dati comunicati per fingersi una o più delle entità coinvolte in un'associazione comunicativa

social engineering ☆ eufemismo per metodi non tecnici e a bassa tecnologia, spesso coinvolgenti trucchi o frode, che sono usati per attaccare i sistemi informativi

phishing ☆ una tecnica per tentare di acquisire dati sensibili, come numeri di conti bancari, attraverso una sollecitazione fraudolenta in un messaggio di posta elettronica o su un sito Web, in cui il perpetratore si finge una persona rispettabile o d'affari legittima

pharming ✤ attacco mirato a reindirizzare il traffico di un sito Web a un altro sito Web fraudolento

malware un qualsiasi software contenente codice malevolo

virus malware che, se eseguito dall'utente, provoca danni nel sistema; non è in grado di propagarsi ad altri sistemi, a meno che l'utente non trasferisce, anche involontariamente, il file infetto

worm malware che satura le risorse della CPU, della memoria o di rete del sistema creando delle repliche di se stesso, ed è in grado di propagarsi ad altri sistemi senza l'intervento dell'utente

cavallo di Troia ✡ il mezzo attraverso cui i malware vengono distribuiti solitamente

backdoor ✡ punto di accesso non autorizzato che può essere usato da un malware per aggirare il sistema di autenticazione ed entrare nel sistema

rootkit ✡ insieme di strumenti, nascosti nel sistema, per ottenere l'accesso al sistema con privilegi di amministratore (ad es. programma modificato, libreria, driver, modulo kernel, hypervisor)

A.2 Concetti base di sicurezza informatica

crittografia ✪ la disciplina che incorpora i principi, i mezzi e i metodi per la trasformazione dei dati al fine di nascondere il loro contenuto informativo [...]

criptazione ✪ la trasformazione crittografica dei dati per produrre il testo cifrato

cifrario ☆ un algoritmo crittografico per la criptazione e la decriptazione

testo in chiaro ✪ dati intelligibili, il cui contenuto semantico è disponibile

testo cifrato ✪ dati prodotti attraverso l'uso della cifratura; il contenuto semantico dei dati risultanti non è disponibile

chiave ☆ un parametro in ingresso usato per variare una funzione di trasformazione effettuata da un algoritmo crittografico

crittografia simmetrica ☆ una branca della crittografia in cui gli algoritmi usano la stessa chiave per entrambe le due operazioni crittografiche delle controparti (ad es. criptazione e decriptazione)

chiave simmetrica ☆ una chiave crittografica che è usata in un algoritmo crittografico simmetrico

crittografia asimmetrica ☆ una moderna branca della crittografia (nota popolarmente come "crittografia a chiave pubblica") in cui gli algoritmi usano una coppia di chiavi (una chiave pubblica e una chiave privata) e usano un componente diverso della coppia per ognuna delle due operazioni crittografiche delle controparti (ad es. criptazione e decriptazione, o creazione della firma e verifica della firma)

chiave asimmetrica ☆ una chiave crittografica che è usata in un algoritmo crittografico asimmetrico

coppia di chiavi ☆ un insieme di chiavi correlate matematicamente – una chiave pubblica e una chiave privata – che sono usate per la crittografia asimmetrica e sono generate in un modo che rende computazionalmente infattibile derivare la chiave privata dalla conoscenza della chiave pubblica

chiave pubblica ☆ la componente pubblicamente divulgabile di una coppia di chiavi crittografiche usata per la crittografia asimmetrica

chiave privata ☆ la componente segreta di una coppia di chiavi crittografiche usata per la crittografia asimmetrica

resistente ☆ usato per descrivere un algoritmo crittografico che richiederebbe una grande quantità di potenza computazionale per sconfiggerlo

lunghezza della chiave ☆ il numero di simboli (di solito dichiarato come un numero di bit) necessario per poter rappresentare uno qualsiasi dei valori possibili di una chiave crittografica

lunghezza della chiave efficace ★ una misura della resistenza di un algoritmo crittografico, indipendentemente dalla lunghezza effettiva della chiave

security by obscurity ☆ tentare di mantenere o aumentare la sicurezza di un sistema tenendo segrete la progettazione o la costruzione di un meccanismo di sicurezza

key establishment ☆ una procedura che combina le fasi di generazione della chiave e di distribuzione della chiave necessarie per instaurare o installare un'associazione comunicativa sicura

generazione della chiave ☆ un processo che crea la sequenza di simboli che compongono una chiave crittografica

distribuzione della chiave ☆ un processo che consegna una chiave crittografica dalla posizione dove viene generata alle posizioni dove viene usata in un algoritmo crittografico

out-of-band ☆ trasferimento di informazioni utilizzando un canale o un metodo che è fuori da (ad es. separato da o diverso da) il canale principale o il metodo normale

trasporto della chiave ☆ un metodo di key establishment tramite il quale una chiave segreta viene generata da un'entità di sistema in un'associazione comunicativa e inviata in modo sicuro a un'altra entità nell'associazione

key agreement ☆ un metodo per negoziare un valore di chiave in linea senza trasferire la chiave, neanche in forma criptata, ad es. la tecnica Diffie-Hellman

cifrario a blocchi ☆ un algoritmo di criptazione che rompe il testo in chiaro in segmenti di dimensione fissa e usa la stessa chiave per trasformare ciascun segmento in chiaro in un segmento di testo cifrato di dimensione fissa

modalità operativa ☆ una tecnica per migliorare l'effetto di un algoritmo crittografico o per adattare l'algoritmo per un'applicazione, come applicare un cifrario a blocchi a una sequenza di blocchi di dati o a un flusso di dati

attacco known-plaintext ☆ una tecnica di crittoanalisi in cui l'analista prova a determinare la chiave dalla conoscenza di alcune coppie testo in chiaro-testo cifrato (anche se l'analista potrebbe anche avere altri indizi, come la conoscenza dell'algoritmo crittografico)

valore di inizializzazione (IV) ☆ un parametro in ingresso che imposta lo stato di partenza di una modalità o algoritmo crittografico

liveness ☆ una proprietà di un'associazione comunicativa o una funzionalità di un protocollo di comunicazione che fornisce la garanzia al destinatario dei dati che i dati stanno venendo trasmessi in modo fresco dal suo originatore, ad es. che i dati non stanno venendo ripetuti, da parte dell'originatore o di un terzo, da una trasmissione precedente

nonce ☆ un valore casuale o che non si ripete che è incluso nei dati scambiati da un protocollo, di solito con lo scopo di garantire la liveness e quindi di rilevare e proteggere da attacchi replay

cifrario di tipo stream ★ un algoritmo di criptazione che rompe il testo in chiaro in un flusso di elementi successivi (solitamente bit) e cripta l'n-esimo elemento del testo in chiaro con l'n-esimo elemento di un flusso parallelo di chiave, convertendo così il flusso in chiaro in un flusso cifrato

one-time pad ☆ un algoritmo di criptazione in cui la chiave è una sequenza casuale di simboli e ogni simbolo viene usato per la criptazione una sola volta – ad es. usato per criptare un solo simbolo in chiaro e produrre così un solo simbolo cifrato – e una copia della chiave viene usata similmente per la decriptazione

pseudocasuale ☆ una sequenza di valori che sembra casuale (ad es. impredicibile) ma è in realtà generata da un algoritmo deterministico

seme ☆ un valore che è un ingresso a un generatore di numeri pseudocasuali

crittografia a curve ellittiche (ECC) ★ un tipo di crittografia asimmetrica basata sulla matematica dei gruppi che sono definiti dai punti su una curva, dove la curva è definita da un'equazione quadratica in un campo finito

funzione di hash ☆ una funzione H che mappa una stringa di bit arbitraria a lunghezza variabile, s, in una stringa a lunghezza fissa, $h = H(s)$ [...]

digest ✣ una sorta di "riassunto" a lunghezza fissa dei dati, calcolato solitamente per mezzo di un algoritmo di hash crittografico dedicato (= a scopo di sicurezza)

criptosistema l'insieme di un algoritmo di crittografia e di un algoritmo di hash usato per garantire la sicurezza di un certo sistema

keyed hash ☆ un hash crittografico in cui la mappatura a un risultato di hash è variata da un secondo parametro in ingresso che è una chiave crittografica

firma digitale ☆ un valore calcolato con un algoritmo crittografico e associato con un oggetto di dati in modo tale che qualsiasi destinatario dei dati può usare la firma per verificare l'origine e l'integrità dei dati

meccanismo di authenticated encryption ✣ tecnica crittografica usata per proteggere la riservatezza e garantire l'origine e l'integrità dei dati, e che consiste di due processi componenti: un algoritmo di criptazione e un algoritmo di decriptazione

key escrow ✣ possibilità di recuperare una chiave anche senza il consenso del proprietario

key recovery ✦ meccanismi e processi che consentono a parti autorizzate di recuperare la chiave crittografica usata per la riservatezza dei dati

A.3 Lo standard X.509, le PKI ed i documenti elettronici

certificato a chiave pubblica ✣ una struttura dati usata per legare in modo sicuro una chiave pubblica ad alcuni attributi

public-key infrastructure (PKI) ✣ l'infrastruttura tecnica ed organizzativa preposta alla creazione, distribuzione e revoca dei certificati a chiave pubblica

certification authority (CA) ☆ un'entità che emette certificati digitali (soprattutto certificati X.509) e attesta la corrispondenza tra gli elementi di dato in un certificato

registration authority (RA) ☆ un'entità PKI facoltativa (separata dalle CA) che non firma né certificati digitali né CRL ma ha la responsabilità di registrare o verificare alcune o tutte le informazioni (in particolare le identità dei soggetti) necessarie a una CA per emettere i certificati e le CRL e per svolgere altre funzioni di gestione dei certificati

end entity ☆ un'entità di sistema che è il soggetto di un certificato a chiave pubblica e che sta usando, o è autorizzato e in grado di usare, la chiave privata corrispondente solo per scopi diversi dalla firma di un certificato digitale, cioè un'entità che non è una CA

relying party ☆ un'entità di sistema che dipende dalla validità delle informazioni (come il valore della chiave pubblica di un'altra entità) fornite da un certificato digitale

gerarchia di certificazione ☆ una topologia strutturata ad albero (priva di cicli) di relazioni tra le CA e le entità a cui le CA emettono certificati a chiave pubblica

CA radice ☆ la CA che è la CA di livello più alto (la più fidata) in una gerarchia di certificazione, cioè l'autorità sulla cui chiave pubblica tutti gli utenti dei certificati basano la validazione di certificati, CRL, percorsi di certificazione e altri costrutti

certificato self-signed ☆ un certificato a chiave pubblica per cui la chiave pubblica associata dal certificato e la chiave privata usata per firmare il certificato sono componenti della stessa coppia di chiavi, che appartiene al firmatario

emissione ☆ generare e firmare un certificato digitale (o una CRL) e, solitamente, distribuirlo e renderlo disponibile a potenziali utenti del certificato (o utenti della CRL)

soggetto ☆ il nome (di un'entità di sistema) che è associato agli elementi di dato in un certificato digitale; ad es., un DN che è associato a una chiave nel certificato a chiave pubblica

emettitore ☆ la CA che firma un certificato digitale o una CRL

repository ☆ un sistema per archiviare e distribuire i certificati digitali e le relative informazioni (compresi le CRL, i CPS e le politiche di certificato) agli utenti dei certificati

revoca di certificato ☆ l'evento che accade quando una CA dichiara che un certificato digitale precedentemente valido emesso da quella CA è diventato non valido; solitamente dichiarato con una data effettiva

certificate revocation list (CRL) ☆ una struttura dati che enumera i certificati digitali che sono stati invalidati dal loro emettitore prima di quando si pianificava che scadessero

delta CRL ☆ una CRL parziale che contiene solo entry per i certificati che sono stati revocati dall'emissione di una precedente base CRL

indirect certificate revocation list (ICRL) ☆ in X.509, una CRL che può contenere le notifiche di revoca di certificato per certificati emessi da CA diverse dall'emettitore (cioè il firmatario) della ICRL

certificate status responder ☆ un server online fidato che agisce per una CA per fornire le informazioni autenticate sullo stato di un certificato agli utenti di certificato; offre un'alternativa all'emissione di una CRL

time stamp ☆ rispetto a un oggetto di dato, un'etichetta o un marchio in cui è registrato il tempo (ora del giorno o altro istante di tempo trascorso) in cui l'etichetta o il marchio è stato affisso all'oggetto di dato

personal security environment (PSE) ✧ archiviazione locale sicura per la chiave privata di un'entità, la chiave della CA direttamente fidata ed eventualmente altri dati; a seconda della politica di sicurezza dell'entità o dei requisiti di sistema, questo può essere, per esempio, un file protetto crittograficamente o un token hardware resistente alle manomissioni

token crittografico ☆ un dispositivo fisico, controllato dall'utente e portatile (ad es. smart card o carta PCMCIA) usato per memorizzare le informazioni crittografiche ed eventualmente anche svolgere le funzioni crittografiche

smart card crittografica ★ carta a chip con memoria e con capacità crittografiche autonome

protocollo proof-of-possession ☆ un protocollo con il quale un'entità di sistema dimostra a un'altra di possedere e controllare una chiave crittografica o altre informazioni segrete

busta digitale ☆ una combinazione di

(a) dati (di qualsiasi tipo) con contenuto criptato intesi per un destinatario

(b) la chiave di criptazione del contenuto in forma criptata che è stata preparata per l'uso da parte del destinatario

firma elettronica ★ l'insieme dei dati in formato elettronico che sono attaccati o logicamente associati con altri dati in formato elettronico e che ne forniscono un mezzo di autenticazione

qualified certificate ✯ un certificato a chiave pubblica che ha lo scopo primario di identificare una persona con un livello di certezza elevato, dove il certificato soddisfa alcuni requisiti di qualificazione definiti da un quadro giuridico applicabile, come la Direttiva Europea sulla Firma Elettronica

A.4 Sistemi di autenticazione

autenticazione ☆ il processo di verifica di una dichiarazione che un'entità di sistema o una risorsa di sistema ha un certo valore di attributo

identificazione ☆ un atto o processo che presenta un identificativo a un sistema in modo che il sistema possa riconoscere un'entità di sistema e distinguerla dalle altre entità

verifica ☆ il processo di esaminamento delle informazioni per stabilire la verità di un dato o valore dichiarato

attacco a dizionario ☆ un attacco che usa una tecnica a forza bruta provando in successione tutte le parole in una grande lista esaustiva

ransomware ✯ un tipo di malware che limita l'accesso al sistema di elaborazione che infetta, e chiede un riscatto pagato al creatore/i del malware affinché la restrizione venga rimossa

sale ☆ un valore di dato usato per variare i risultati di un calcolo in un meccanismo di sicurezza, in modo che un risultato computazionale esposto di una istanza di applicazione del meccanismo non possa essere riutilizzato da un attaccante in un'altra istanza

a sfida ☆ un processo di autenticazione che verifica un'identità richiedendo che siano fornite le informazioni di autenticazione corrette in risposta a una sfida [...]

password usa e getta ☆ una semplice tecnica di autenticazione in cui ogni password è usata solo una volta come informazione di autenticazione che verifica un'identità [...]

token ☆ un oggetto di dato o un dispositivo fisico usato per verificare un'identità in un processo di autenticazione

autenticazione biometrica ☆ un metodo di generazione delle informazioni di autenticazione di una persona digitalizzando le misure di una caratteristica fisica o comportamentale [...]

Single Sign-On (SSO) ✱ fornire all'utente un'unica "credenziale" con cui autenticarsi per tutte le operazioni su qualunque sistema

A.5 Sicurezza delle reti IP

accounting ✲ il processo di tenere traccia e/o analizzare le attività degli utenti su una rete registrando i dati chiave (ad es. quantità di tempo nella rete, servizi acceduti, quantità di dati trasferiti)

Virtual Private Network (VPN) ✳ una tecnica hardware e/o software per realizzare una rete privata utilizzando canali e apparati di trasmissione condivisi o comunque non fidati

A.6 Firewall e IDS/IPS

firewall ☆ un gateway di internetwork che limita il traffico di comunicazione dati da e verso una delle reti connesse (quella che si dice "dentro" il firewall) e così protegge le risorse di sistema di quella rete da minacce dall'altra rete (quella che si dice "fuori" dal firewall)

bastion host ☆ un computer fortemente protetto che sta in una rete protetta da un firewall (o è parte di un firewall) ed è l'unico host (o uno di solo alcuni) nella rete che può essere acceduto direttamente dalle reti dall'altra parte del firewall

zona buffer ☆ un segmento di Internetwork neutrale usato per connettere altri segmenti che operano ciascuno sotto una diversa politica di sicurezza

honey pot ☆ un sistema (ad es. un server Web) o una risorsa di sistema (ad es. un file su un server) che è progettato per attirare potenziali cracker e intrusi, come il miele è appetitoso per gli orsi

screening router ☆ un router Internetwork che impedisce selettivamente il passaggio di pacchetti di dati secondo una politica di sicurezza

rilevamento delle intrusioni ☆ sentire e analizzare eventi di sistema allo scopo di notare (cioè diventare consapevoli di) tentativi di accedere alle risorse del sistema in una maniera non autorizzata

Intrusion Detection System (IDS) ✳ sistema per identificare individui che usano un computer o una rete senza autorizzazione

Intrusion Prevention System (IPS) ✦ sistema che può rilevare un'attività intrusiva e può anche provare a fermare l'attività, idealmente prima che essa raggiunga i suoi obiettivi

Fonti

Diapositive del corso

✡ *Introduzione alla sicurezza delle reti e dei sistemi informativi* © Antonio Lioy - Politecnico di Torino (1995-2014)

✣ *Concetti base di sicurezza informatica* © Antonio Lioy - Politecnico di Torino (1995-2014)

★ *Lo standard X.509, le PKI ed i documenti elettronici* © Antonio Lioy - Politecnico di Torino (1997-2011)

✱ *Sistemi di autenticazione* © Antonio Lioy - Politecnico di Torino (1995-2014)

✳ *Sicurezza delle reti IP* © Antonio Lioy - Politecnico di Torino (1997-2014)

✶ *Firewall e IDS/IPS* © Marco Domenico Aime, Antonio Lioy - Politecnico di Torino (1995-2014)

Request for Comments

☆ RFC 4949: R. Shirey, *Internet Security Glossary, Version 2* © The IETF Trust (2007)

✯ RFC 3739: S. Santesson, M. Nystrom, T. Polk, *Internet X.509 Public Key Instrastructure: Qualified Certificates Profile* © The Internet Society (2004)

ISO International Standard

✪ ISO 7498-2:1989: *Information processing systems – Open Systems Interconnection – Basic Reference Model – Part 2: Security Architecture* © 1989 ISO

✢ ISO/IEC 9594-8:1998: *Information Technology – Open Systems Interconnection – The Directory: Authentication framework* © 1998 ISO/IEC

✧ ISO/IEC 15945:2002: *Information technology – Security techniques – Specification of TTP services to support the application of digital signatures* © 2002 ISO/IEC

✣ ISO/IEC 19772:2009: *Information technology – Security techniques – Authenticated encryption* © 2009 ISO/IEC

✤ ISO/IEC 29115:2013: *Information technology – Security techniques – Entity authentication assurance framework* © 2013 ISO/IEC

Altro

☆ G. Huff, *Trusted Computer Systems – Glossary*, MTR 8201, The MITRE Corporation, March 1981

✯ *2014 Data Breach Investigations Report* (DBIR) © 2014 Verizon

✯ B. Schneier, *Applied Cryptography Second Edition*, John Wiley & Sons, Inc., New York, 1996

✯ NSA, *Information Assurance Technical Framework*, Release 3, NSA, settembre 2000

✦ CNSSI-4009: The Committee on National Security Systems Instruction No 4009 *National Information Assurance Glossary* © NCSC

☆ Wikipedia

✲ Portale linguistico Microsoft © Microsoft