



POLITECNICO DI TORINO

Sicurezza dei sistemi informativi

Appunti dal corso 01TYMOV del Prof. Antonio Lioy

A.A. 2020/21

Autore: Marco Smorti

Revisore: Riccardo Zaccone

Contenuti aggiornati alla lezione: 20 (11/12/2020)

Revisione dei contenuti aggiornata alla lezione: 20

(11/12/2020) Ultima modifica il: 20/01/2021 13:13:00

AVVISO IMPORTANTE: questi appunti non sono in alcun modo promossi o controllati dal professore o da qualsiasi persona del corpo docente del corso, pertanto sono forniti "COSÌ COME SONO" e senza alcuna garanzia. Tuttavia, ci siamo impegnati a fondo per realizzarli e verificarne la correttezza, speriamo che vi siano utili.

Sintesi

Introduzione alla sicurezza dei sistemi ICT	7
Complessità dello scenario ICT.....	7
Una definizione di sicurezza ICT	7
Stima del rischio.....	9
Analisi e gestione della sicurezza.....	10
Relazioni nel campo della sicurezza.....	11
Finestra di esposizione.....	12
Che cos'è la sicurezza?	14
Principi di sicurezza.....	14
Proprietà di sicurezza.....	15
Piramide della sicurezza	16
Protezione dei dati	17
Alcune classi di attacchi	19
Sniffing dei pacchetti (intercettazione).....	20
Analisi del traffico	20
Spoofing IP (masquerading).....	20
Negazione del servizio (DoS).....	20
Negazione di servizio distribuita (DDOS)	21
Server ombra	22
Dirottamento della connessione / Man In The Middle (MITM)	22
Trojan / Uomo nel browser (MITB).....	23
Zeus	23
Bug del software	23
Virus & Co. (malware).....	23
Virus e worm (malware).....	24
Catena alimentare del malware	24
Zeus	24
Ransomware	25
Ransomware-as-a-service	25
Problemi di base (non tecnologici)	26
Ingegneria sociale	27
Alcuni importanti attacchi recenti	29
Contro i sistemi cyber-fisici	30
Contro le infrastrutture critiche	31
Contro Internet-of-Things, automotive, casa.....	32
Fondamenti di sicurezza ICT	35
Chiave segreta / crittografia simmetrica.....	36

Algoritmi simmetrici	36
Applicazione degli algoritmi a blocchi.....	38
BCE (Libro dei codici elettronici).....	39
.....	39
CBC (Cipher Block Chaining)	39
Imbottitura (allineamento, riempimento)	40
Furto del testo cifrato (CTS).....	41
CTR (modalità contatore)	42
Algoritmi di tipo stream	43
Salsa20 e ChaCha20	43
Crittografia simmetrica.....	44
Distribuzione delle chiavi per la crittografia simmetrica	44
Le sfide del DES.....	45
Crittografia a chiave pubblica / asimmetrica	46
Firma digitale	46
Riservatezza senza segreti condivisi	46
Algoritmi a chiave pubblica	46
Ottimizzazione computazionale RSA.....	47
Scintillio	47
Twirl (il localizzatore di relazioni dell'Istituto Weizmann).....	48
Firma del firmware per le calcolatrici TI.....	48
Distribuzione delle chiavi per la crittografia asimmetrica.....	48
Scambio di chiavi segrete con algoritmi asimmetrici.....	48
Algoritmo Diffie-Hellman	49
Integrità del messaggio	51
Funzioni hash crittografiche	51
KDF (funzione di derivazione delle chiavi)	53
Autenticazione tramite digest a chiave.....	54
Crittografia autenticata	56
Autenticazione tramite digest e crittografia asimmetrica: firma digitale	59
Certificato a chiave pubblica	60
PKI (Infrastruttura a chiave pubblica)	61
Soluzioni suggerite per la sicurezza	64
Problemi di sicurezza nei prodotti importati: una retrospettiva.....	64
Tecniche e architetture di autenticazione	67
Modello di autenticazione digitale (NIST SP800.63B).....	67
Autenticazione dell'utente.....	68
Autenticazione forte (tra pari)	72

Autenticazione a risposta multipla (CRA).....	73
Sistemi (simmetrici) sfida-risposta.....	73
Sistemi di sfida-risposta (asimmetrici).....	74
Password unica (OTP).....	75
OTP a tempo.....	77
OTP basato su eventi	79
OTP fuori banda	79
AuthN a due/multi fattori (2FA/MFA).....	79
Autenticazione degli esseri umani.....	80
Interoperabilità dell'autenticazione: OATH	81
FIDO.....	82
Sicurezza delle reti IP.....	86
Autenticazione dei canali PPP.....	86
Autenticazione di un accesso alla rete.....	87
EAP.....	87
RADIO	89
DIAMETRO	92
IEEE 802.1x	92
Implementazione della sicurezza nei livelli OSI	94
Sicurezza a livello di rete (L3)	95
Che cos'è una VPN?	95
VPN tramite indirizzi privati	95
VPN tramite tunnel.....	96
VPN tramite tunnel IP	96
VPN tramite tunnel IP sicuro	96
IPsec	97
Associazione di sicurezza (SA) IPsec	97
Modalità di trasporto IPsec.....	98
Modalità tunnel IPsec	99
AH	99
ESP	101
Dettagli sull'implementazione di IPsec.....	102
Protezione IPsec replay (parziale)	102
IPsec v3	103
Modi di utilizzare IPsec.....	104
Gestione delle chiavi IPsec	105
"Protocolli di sicurezza "Servizio	107
Sicurezza DNS.....	110

Traduzione nome-indirizzo.....	111
Sicurezza delle applicazioni di rete	114
SSL (Secure Socket Layer)	115
TLS	116
ID sessione	118
TLS e server virtuali: il problema.....	120
TLS e server virtuali: soluzioni	120
Estensione ALPN (Application-Layer Protocol Negotiation).....	120
DTLS	121
Il problema del downgrade di TLS.....	121
TLS fallback Valore della suite di cifratura di segnalazione (SCSV).....	121
Sicurezza HTTP	122
Autenticazione HTTP digest.....	122
HTTP e SSL/TLS	123
Sicurezza del trasporto HTTP (HSTS).....	124
HTTP Public Key Pinning (HPKP).....	125
Sistemi di pagamento elettronico	126
Firewall e IDS/ISP	128
Cos'è un firewall	128
Design del firewall.....	128
Componenti di base di un firewall.....	129
"Architettura "Screening Router	129
"Architettura "dual-homed gateway	130
"Architettura "ospite schermato	130
"Architettura della "sottorete schermata	131
Tecnologie firewall	132
Protezione offerta da un firewall	136
Sistema di rilevamento delle intrusioni (IDS)	136
IPS - Sistema di prevenzione delle intrusioni.....	137
Firewall di nuova generazione (NGFW)	137
Gestione unificata delle minacce (UTM)	138
Vaso di miele/rete di miele	138
Sicurezza della posta elettronica.....	138
MHS (Sistema di gestione dei messaggi)	138
Posta elettronica in modalità client-server	139
Webmail	139
Protocolli, porte e formati.....	139
Messaggi RFC-822	140

Spamming della posta.....	142
Relè di posta (aperto).....	142
Antispam per MSA.....	143
Anti-spam per l'MTA in arrivo	143
ESMTP	144
AUTH: metodi di sfida-risposta	145
Protezione di SMTP con TLS.....	146
Servizi di sicurezza per i messaggi di posta elettronica	147
Sicurezza della posta elettronica - idee principali	147
Tipi di messaggi sicuri.....	147
Messaggi sicuri: creazione.....	148
Formati di posta elettronica sicuri	148
PGP (Pretty Good Privacy).....	148
MIME (Multipurpose Internet Mail Extensions).....	149
Servizi di posta elettronica client-server.....	153

Introduzione alla sicurezza dei sistemi ICT

La cybersecurity oggi è diventata molto importante: poiché ogni sistema è costruito su sistemi informatici, qualsiasi tipo di danno costerà quasi una perdita economica, e questo è il problema principale. Anche gli attacchi indiretti, che non mirano a rubare denaro, hanno costi economici. La sicurezza informatica è importante anche perché l'attacco può essere effettuato senza recarsi fisicamente sul luogo dell'obiettivo.

I motivi per cui la Cybersecurity è importante sono:

- Grossi danni in caso di attacchi riusciti;
- Facile accessibilità dei sistemi;

Dobbiamo considerare tutte le possibili conseguenze di un attacco riuscito. In primo luogo, ci possono essere **perdite finanziarie** (*perdite dirette* se, ad esempio, ottengo le credenziali dei conti bancari e *perdite indirette* se l'informazione che l'azienda è stata attaccata ha effetti negativi sulla borsa). Ci possono essere **costi di recupero**, perché ogni attacco riuscito è un danno, e ci saranno costi per riportare il sistema alle normali operazioni e per migliorare il sistema per evitare nuovi attacchi. Ci può essere una **perdita di produttività** se gli attacchi bloccano o ritardano i processi. Un attacco riuscito può provocare un'**interruzione dell'attività**, perché i clienti possono rivolgersi a fornitori alternativi se un'azienda è facile da attaccare.

Per tutti questi motivi dovremmo proteggere i sistemi. La maggior parte dell'innovazione odierna si basa su due pilastri principali:

- Il fatto che possiamo sempre comunicare da ogni parte del mondo (reti di comunicazione);
- Il crescente utilizzo di dispositivi personali e mobili;

Questi due pilastri non sono più sufficienti per un prodotto innovativo: ogni nuovo prodotto ha bisogno di un sistema di sicurezza.

Complessità dello scenario TIC

Lo scenario delle TIC è complesso per molte ragioni. Ad esempio, la grande quantità di **dispositivi** mobili e connessi: desktop, laptop, tablet, smartphone, smart TV, frigorifero, automobile. Tutti questi dispositivi possono ora comunicare attraverso Internet, quindi devono essere protetti. Anche le **reti di comunicazione** sono diventate reti di soli dati (il che significa che non esiste più una rete telefonica analogica) e tutto passa attraverso la rete. Ciò significa che tutto è attaccabile. Non solo le reti wireless possono essere attaccate, ma anche le reti cablate sono altamente attaccabili e insicure. I **servizi distribuiti** sono in aumento e questo significa che spesso siamo chiamati a trovare soluzioni tecniche per sostenerli, esternalizzando alcune parti del server, dell'hosting, del farming e infine utilizzando il cloud. Questo significa che i computer non sono più all'interno di un'azienda ma da qualche altra parte, quindi è necessario fidarsi anche del provider che ospita i servizi. Inoltre, la programmazione sta diventando sempre più complessa a causa della stratificazione del software, dell'integrazione dei framework, dei mix di linguaggi, e questo significa che è più facile commettere errori.

Per quanto riguarda la sicurezza, queste motivazioni possono essere riassunte nel **primo assioma**

dell'ingegneria: "Più un sistema è complesso, più difficile sarà la sua verifica di correttezza".

Ciò significa che dobbiamo mantenere un sistema il più semplice possibile. Ad esempio, il numero di bug in un programma è più che quadratico rispetto al suo numero di righe di codice e la complessità degli attuali sistemi informatici è a favore degli attaccanti, che possono trovare percorsi di attacco sempre più ingegnosi e imprevisti dai difensori. Un modo per esprimere questo concetto è la **regola KISS**: "Keep it Simple, Stupid".

Una definizione di sicurezza delle TIC

Ognuno di noi ha un concetto diverso di sicurezza: ad esempio, l'obbligo di utilizzare le cinture di sicurezza alla guida dipende da Paese a Paese. La sicurezza è un concetto personale, ma a livello ingegneristico è necessario dare un'impronta formale alla sicurezza.

definizioni. La cybersecurity è una parte distribuita di un'azienda e ogni dipendente di un'azienda deve avere la consapevolezza della cybersecurity.

1. "La *cybersecurity* è l'insieme di prodotti, servizi, regole organizzative e comportamenti individuali che proteggono il sistema ICT di un'azienda".

Spieghiamo le parole chiave della definizione.

- **Prodotti:** si riferisce a qualcosa che le persone possono acquistare (come i prodotti per firewall e VPN);
 - **Servizi:** questi servizi sono implementati dall'acquisto di prodotti;
 - **Regole organizzative:** sono necessarie perché anche se, ad esempio, viene creato un nuovo sistema con password, le regole devono dare informazioni ai dipendenti su quanto deve essere complessa la password, altrimenti non ci saranno regole ma comportamenti personali, che potrebbero rendere meno efficace l'uso di soluzioni tecniche;
2. "*È il dovere di proteggere le risorse da accessi indesiderati, garantire la privacy delle informazioni, assicurare il funzionamento del servizio e la disponibilità in caso di eventi imprevedibili (C.I.A. = Confidenzialità, Integrità, Disponibilità)*".

Più in dettaglio, questi tre concetti hanno il seguente significato:

- **La riservatezza** comprende due concetti correlati:
 - **Riservatezza dei dati:** assicura che le informazioni private o riservate non siano rese disponibili o divulgiate a persone non autorizzate;
 - **Privacy:** Garantisce che gli individui controllino o influenzino le informazioni che li riguardano che possono essere raccolte e conservate e da chi e a chi tali informazioni possono essere divulgate.

In termini di requisiti e di definizione di perdita di sicurezza, significa preservare le restrizioni autorizzate all'accesso e alla divulgazione delle informazioni, compresi i mezzi per proteggere la privacy personale e le informazioni proprietarie. Una perdita di riservatezza è la divulgazione non autorizzata di informazioni.

- **L'integrità** comprende due concetti correlati:
 - **Integrità dei dati:** assicura che le informazioni (sia memorizzate che nei pacchetti trasmessi) e i programmi vengano modificati solo in modo specifico e autorizzato;
 - **Integrità del sistema:** assicura che un sistema svolga la funzione prevista in modo ineccepibile, senza manipolazioni non autorizzate, intenzionali o involontarie, del sistema.

In termini di requisiti e di definizione di perdita di sicurezza, significa prevenire la modifica o la distruzione impropria delle informazioni, garantendo anche la non ripudiabilità e l'autenticità delle informazioni. Una perdita di integrità è la modifica o la distruzione non autorizzata delle informazioni.

- **Disponibilità:** assicura che i sistemi funzionino tempestivamente e che il servizio non sia negato agli utenti autorizzati. In termini di requisiti e di definizione di perdita di sicurezza, significa garantire un accesso e un utilizzo tempestivo e affidabile delle informazioni. Una perdita di disponibilità è l'interruzione dell'accesso o dell'uso delle informazioni o di un sistema informativo.

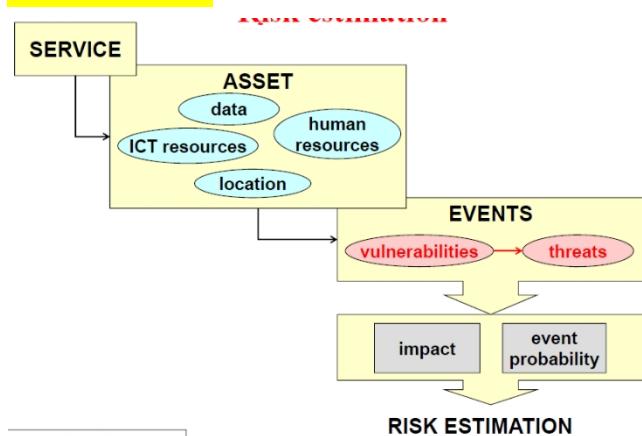
Questa definizione è un buon punto di partenza per la cybersecurity, ma è necessario qualcosa di più della triade C.I.A.; due delle più citate sono le seguenti:

- **Autenticità:** la proprietà di essere autentici e di poter essere verificati e fidati; fiducia nella validità di una trasmissione, di un messaggio o dell'originatore di un messaggio. Ciò significa verificare che gli utenti siano chi dicono di essere e che ogni input che arriva al sistema provenga da una fonte affidabile.

- **Accountability:** l'obiettivo di sicurezza che genera il requisito che le azioni di un'entità siano riconducibili in modo univoco a tale entità. Ciò supporta il non ripudio, la deterrenza, l'isolamento dei guasti, il rilevamento e la prevenzione delle intrusioni, il recupero dopo l'azione e le azioni legali.

1. "L'obiettivo è quello di custodire le informazioni con la stessa professionalità e attenzione con cui si custodiscono i gioielli e i certificati di deposito in un caveau bancario".
3. "Il sistema ICT è la cassaforte delle nostre informazioni più preziose; la sicurezza ICT è l'equivalente delle serrature, delle combinazioni e delle chiavi necessarie per proteggerle".

Rischio stima



Prima di impostare una difesa, dobbiamo capire quali sono i rischi. Per fare una stima dei rischi è bene partire dal **servizio**. Una volta conosciuto il servizio che dobbiamo proteggere, dobbiamo identificare quali sono gli asset utilizzati per fornire quel servizio e ci sono 4 categorie di asset: **Risorse ICT** (computer, dischi, reti), **dati** (non i dischi, ma qualcosa di intangibile che potrebbe essere cancellato o modificato), **ubicazione** (gli asset devono essere all'interno di una stanza protetta), **risorse umane** (si intende il numero ristretto di persone che hanno le conoscenze che non devono essere condivise). Dopo aver pensato agli asset, il

Il passo successivo sono gli eventi che potrebbero compromettere il loro normale funzionamento. Il primo punto è che ogni asset ha alcune **vulnerabilità** (ad esempio i dischi, che sono vulnerabili se il martello colpisce il disco) e alcune vulnerabilità possono essere una **minaccia reale** a seconda dell'ambiente. Ad esempio, se il disco viene lasciato in un luogo aperto, qualcuno potrebbe usare il martello per distruggerlo, ma se il disco è chiuso a chiave in una stanza in cui nessuno vi accede, la vulnerabilità esiste ancora, ma non è una minaccia reale.

Quindi, il processo di analisi di un servizio alla ricerca di rischi si svolge come segue:

- Individuare gli **asset** del servizio da proteggere;
- Individuare le **vulnerabilità** di ogni **asset**;
- Una volta identificate le minacce, è necessario decidere per ogni minaccia quale **impatto** potrebbe avere (cosa succede se il disco viene distrutto? Se c'è una sola copia potrebbe essere un disastro, ma se ci sono molte copie non è un problema);
- la **probabilità di evento** della minaccia. Questo è l'ultimo punto per ottenere la

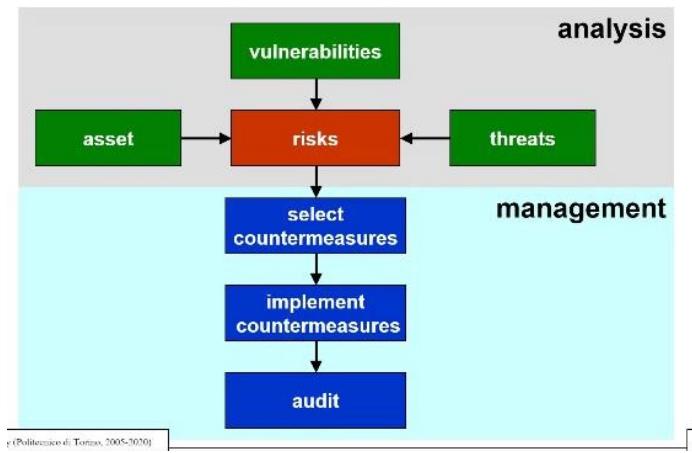
stima del rischio. Alla fine del processo abbiamo un elenco di elementi con il seguente formato:

Nome del	rischioImpatto (ad es. basso, medio, alto)	Probabilità
----------	--	-------------

Ripasso della terminologia:

- **Asset:** l'insieme di beni, dati e persone necessari per un servizio IT;
- **Vulnerabilità:** debolezza di un bene;
- **Minaccia:** azione deliberata/evento accidentale che può produrre la perdita di una proprietà di sicurezza sfruttando una vulnerabilità;
- **Attacco:** minaccia (azione deliberata);
- **(negativo) Evento:** evento di minaccia (evento accidentale);

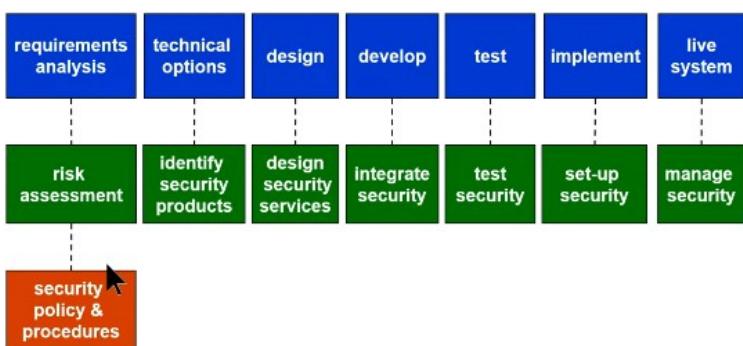
Analisi e gestione della sicurezza di



La stima del rischio è solo un tassello per l'*analisi e la gestione della sicurezza*. Possiamo vedere che gli **asset**, le **vulnerabilità** e le **minacce** ci danno i **rischi** e quindi iniziamo la **gestione della sicurezza**. Ciò significa che per quei rischi che non sono accettabili, perché hanno un impatto elevato o un'alta probabilità di finire in un attacco, dobbiamo selezionare le contromisure e implementarle. L'ultima fase è l'**audit**, ovvero una persona indipendente viene a controllare il nostro lavoro (se abbiamo identificato correttamente i rischi, selezionato le contromisure corrette e le abbiamo implementate correttamente).

In questo corso prenderemo in considerazione tre di questi blocchi: le vulnerabilità, le contromisure disponibili e come implementare le contromisure.

Qual è la fase corretta del ciclo di vita di un sistema in cui implementare la sicurezza? Risposta breve: non esiste un punto corretto in cui implementare la sicurezza, perché questa deve essere curata in **ogni fase della progettazione**.



Più in dettaglio, quando eseguiamo **l'analisi dei requisiti** per il nostro sistema dobbiamo eseguire la valutazione dei rischi: sulla base di questi rischi possiamo definire le **politiche e le procedure di sicurezza** che applicheremo per il resto della progettazione del sistema.

Quando valutiamo le **opzioni tecniche**, dobbiamo anche identificare i prodotti di **sicurezza**. Ad esempio, quando si valuta quale database utilizzare, tra le altre cose

Se si considerano caratteristiche come la velocità e il costo, è necessario considerare anche la sicurezza. Se scegliamo un database che critta automaticamente i dati, abbiamo già risolto un problema di sicurezza. Al contrario, se scegliamo un database più veloce ma che non fornisce alcun sistema di crittografia, dovremo progettarlo separatamente con costi e sforzi aggiuntivi.

Quando **progettiamo** i servizi che il sistema offrirà, dobbiamo progettare anche la parte relativa ai servizi di sicurezza. Dobbiamo aggiungere la sicurezza in ogni fase della progettazione e non "in un secondo momento", perché se realizziamo un prototipo di sito web o di app senza alcun sistema di sicurezza, sarà difficile aggiungere sistemi di sicurezza in seguito.

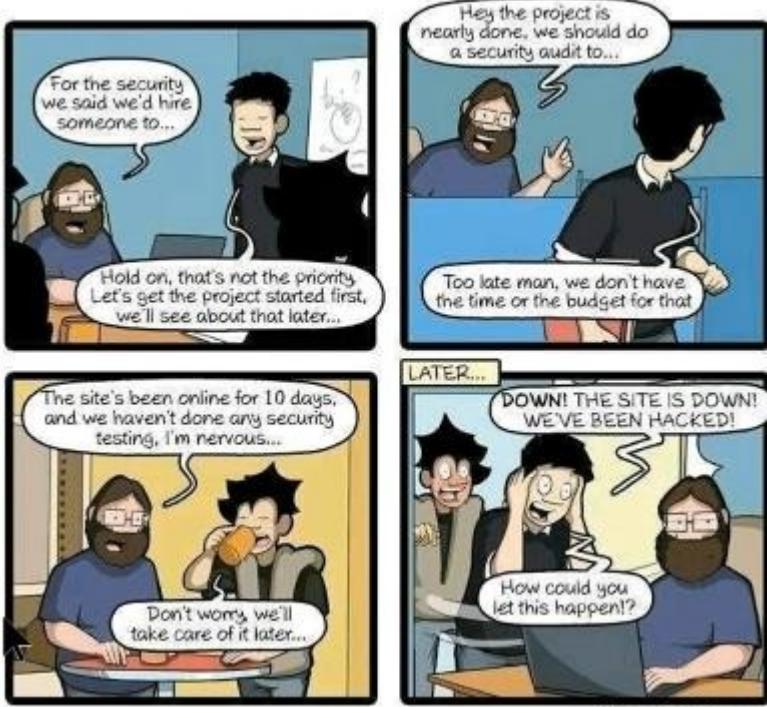


Figura 1 Esempio di fumetto che spiega la realtà dei sistemi

Relazioni nel campo della sicurezza

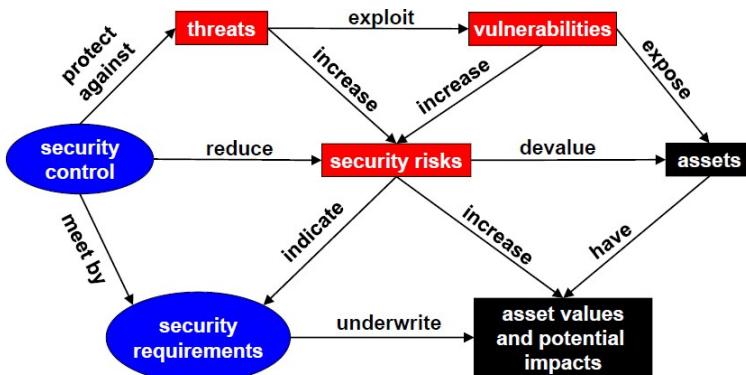


Figura 2 Relazioni nel campo della sicurezza

elemento inserito nel sistema che protegge da una minaccia specifica e riduce i rischi a cui il sistema è esposto. Esempi di controlli di sicurezza sono firewall, VPN e crittografia del disco.

Alcuni termini

- **Incidente:** evento di sicurezza che compromette l'integrità, la riservatezza o la disponibilità di un asset informativo (definizione generica).
- **Violazione (di dati):** incidente che comporta la divulgazione o la potenziale esposizione di dati;
 - **divulgazione:** fornisco dati a qualcuno;
 - **esposizione:** i dati sono disponibili per chiunque sia in grado di sapere dove si trovano;
- **Divulgazione (di dati):** una violazione per la quale è stato confermato che i dati sono stati

Durante lo **sviluppo** del nostro sistema, dobbiamo integrare la sicurezza in ogni fase e per verificare che il progetto sia implementato correttamente dobbiamo **testare il sistema e testare la sicurezza**: un esempio è il test contro gli input inattesi, per essere sicuri che il sistema non accetti ed elabori input sbagliati.

Durante l'**implementazione** del nostro sistema è importante impostare i meccanismi di sicurezza: ci sono diversi sistemi che includono funzioni di sicurezza, ma sono disattivati, perché hanno un costo (in genere attivando la sicurezza il sistema diventa più lento). Ad esempio, la password di default di un router domestico potrebbe essere condivisa tra diverse unità, quindi è necessario cambiarla.

Durante il **funzionamento del sistema**, la sicurezza deve essere gestita giorno per giorno, perché lo scenario della sicurezza cambia ogni giorno.

La scatola nera è il sistema stesso. Le risorse sono esposte a vulnerabilità e tali vulnerabilità aumentano i rischi per la sicurezza. Oltre alle minacce che sfruttano le vulnerabilità, l'esistenza di queste ultime offre l'opportunità di creare una minaccia.

A sinistra ci sono i requisiti di sicurezza che vogliamo implementare. I requisiti di sicurezza sono indicati dai rischi per la sicurezza e i requisiti di sicurezza vengono soddisfatti da

sicurezza controllo. Il **controllo della sicurezza** è oggi il pezzo più importante del quadro: è un

effettivamente divulgati (non solo esposti) a una parte non autorizzata.

La differenza tra le ultime due è che l'ultima è una violazione di dati non divulgati.

Finestra di esposizione

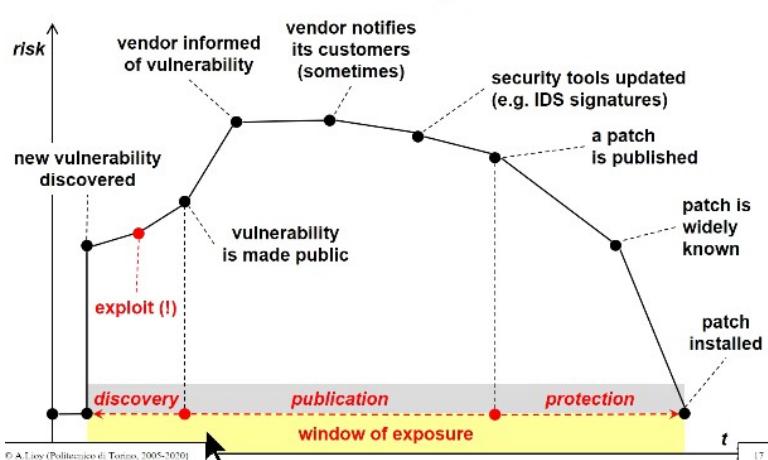
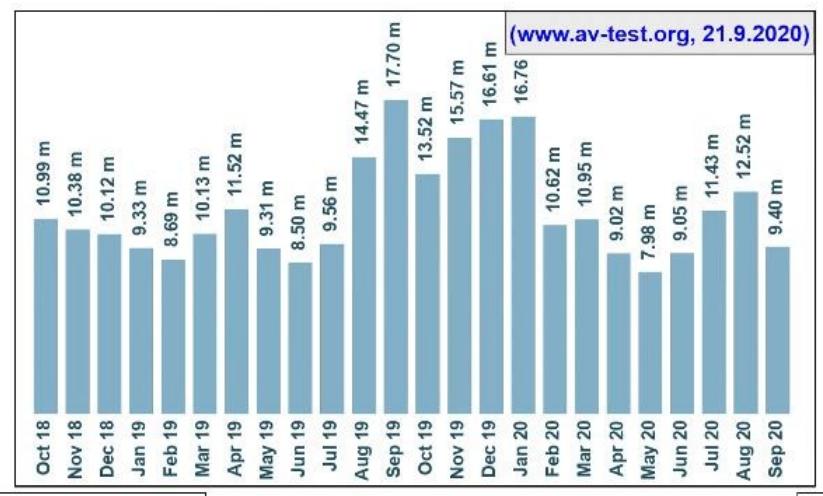


Figura 3 Le tre fasi della finestra di esposizione (WOE): scoperta, pubblicazione e protezione

La finestra di esposizione è il tempo che intercorre tra la scoperta di una nuova vulnerabilità e l'installazione di una patch. Leggendo il grafico dal basso si nota che c'è un livello di rischio basso e non potrà mai essere 0, ma qualcosa di vicino allo 0. A un certo punto viene scoperta una nuova vulnerabilità e il rischio diventa alto (perché se è nuova, non possiamo fermarla). A un certo punto (il punto rosso) qualcuno sfrutta la vulnerabilità per eseguire un attacco. In questo modo la vulnerabilità viene resa pubblica e tutti possono conoscerla. Questa prima fase prende il nome di **scoperta**.

Dopo che l'attacco è stato eseguito, ci sono due categorie di persone informate: i cattivi (che vogliono attaccare il sistema) e i buoni (che cercano di proteggere il sistema). Tra queste due categorie c'è anche il fornitore del prodotto che deve essere informato della vulnerabilità e deve notificare ai suoi clienti la nuova vulnerabilità. Mentre il fornitore lavora per risolvere la vulnerabilità, gli utenti di quel prodotto non dovrebbero cercare di risolverla (in genere è impossibile), ma dovrebbero aggiornare i loro strumenti di sicurezza almeno per rilevare se la vulnerabilità viene utilizzata attivamente per un attacco.

Questa è la fase di **pubblicazione** in cui la vulnerabilità è nota al pubblico e tutti aspettano una correzione e cercano almeno di individuare gli attacchi. Infine, a un certo punto il fornitore crea una patch che risolve la vulnerabilità, ma la patch deve essere distribuita e il rischio diminuisce solo quando la patch è ampiamente conosciuta e finalmente installata. La finestra di esposizione può durare giorni o addirittura mesi: la sicurezza è quindi un lavoro che non finisce mai.



Il grafico mostra che ci sono circa 10 milioni di attacchi al mese che utilizzano malware. Ecco perché dovremmo sempre aggiornare il nostro antivirus/antimalware.

Figura 4 Numero di attacchi mensili al sistema tramite malware



Figura 6 WOE: valore medio per i browser nel periodo 2008-10

Statistiche del valore della finestra di esposizione per i browser. La cosa peggiore è stata nel 2009, quando Safari ha avuto un'esposizione di 13 giorni prima che Apple fosse in grado di fornire una correzione. Inoltre, nel 2008 è stata di 9. Nel 2010 IE ha avuto una media di 4 giorni.

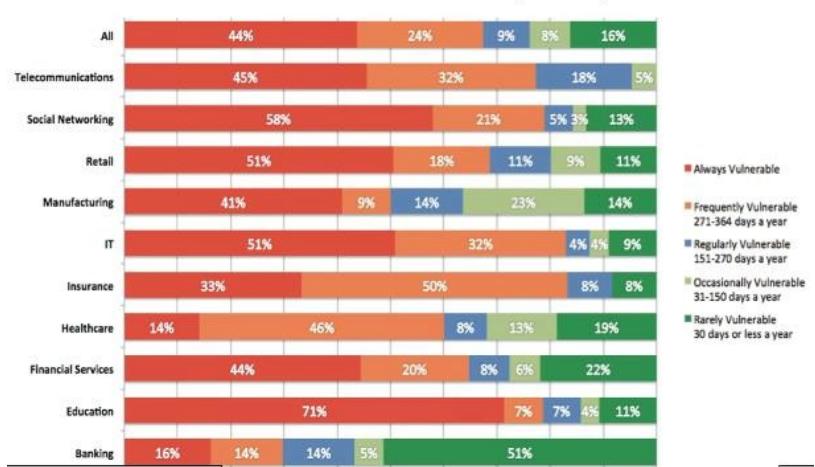


Figura 7 WOE: server web (2010)

Se consideriamo i server web di tutti i settori, il 44% dei server è sempre stato vulnerabile a un attacco in ogni giorno dell'anno.

Il **giorno 0** è letteralmente il primo giorno in cui la vulnerabilità viene segnalata al pubblico (e non è ancora stata risolta).

Il settore bancario è quello con la migliore "barra verde" (raramente vulnerabile, 30 giorni o meno all'anno).

In generale, è difficile avere sicurezza. Le vulnerabilità possono essere scoperte da entrambe le categorie: i buoni e i cattivi. Ci sono persone

che indagano sul software per scoprire le vulnerabilità e informarne il fornitore. Questo viene fatto per correggere le patch prima che la vulnerabilità venga scoperta dai malintenzionati. La "0-day initiative" (ZDI) scopre le vulnerabilità e informa la società prima di renderle pubbliche. La ZDI concede 120 giorni dalla scoperta di una nuova vulnerabilità al fornitore per correggerla prima di renderla pubblica. Termini più lunghi possono essere rischiosi, perché anche i malintenzionati possono scoprire la vulnerabilità nel frattempo.

Esempio:

- 8 maggio 18: ZDI segnala la vulnerabilità al fornitore, il quale conferma la segnalazione.
- 14 maggio 18: il fornitore risponde di aver riprodotto con successo il problema segnalato da ZDI.
- 9 settembre 18: il fornitore ha segnalato un problema con la correzione e che questa *potrebbe non essere rilasciata a settembre*.
- 10 settembre 18: ZDI ha avvertito di un potenziale 0-day (il che significa che la vulnerabilità la cui correzione non è disponibile sta per essere pubblicata).
- 11 settembre 18: il fornitore ha confermato che la correzione non ha reso la compilazione
- 12 settembre 18: ZDI ha confermato l'intenzione di passare allo 0-day il 20 settembre 18.

Cos'è la sicurezza di ?

"La sicurezza è un processo, non un prodotto".

(Bruce Schneier, Crypto-Gram, maggio 2005)

Se abbiamo imparato qualcosa dagli ultimi due anni, è che **le fal当地 nella sicurezza informatica sono inevitabili**. I sistemi si rompono, le vulnerabilità vengono segnalate dalla stampa e ancora molte persone si affidano al prossimo prodotto, al prossimo aggiornamento o alla prossima patch. "Questa volta è sicuro", dicono. Finora non è stato così.

La sicurezza è un processo, non un prodotto. I prodotti forniscono una certa protezione, ma l'unico modo per operare efficacemente in un mondo insicuro è mettere in atto processi che riconoscano l'insicurezza intrinseca dei prodotti. **Il trucco è ridurre il rischio di esposizione indipendentemente dai prodotti o dalle patch.**

Per renderlo possibile è necessario seguire alcuni principi di sicurezza.

Principi di sicurezza

- **Sicurezza in profondità:** se il nemico riesce a sconfiggere la prima linea, deve esserci una seconda linea per fermare l'attaccante. Non affidatevi a una sola difesa, perché potrebbe avere un bug o un problema. È meglio avere più livelli di difesa, in modo che, man mano che l'attaccante sfonda le difese, diventi sempre più difficile continuare a penetrare;
- **Security by design:** significa che la progettazione della sicurezza viene fatta insieme al sistema e non alla fine;
- **Sicurezza predefinita:** gli utenti non dovrebbero avere la possibilità di scegliere se attivare o meno la sicurezza. La sicurezza dovrebbe essere attiva per impostazione predefinita e dovrebbe essere un grosso sforzo eliminare la sicurezza dal sistema.
- **Minimo privilegio:** significa che a ogni elemento che opera all'interno del sistema deve essere assegnata la quantità minima di privilegi che gli consente di svolgere il proprio compito. Immaginiamo di avere un privilegio enorme e che il computer venga attaccato da un virus: il virus potrebbe avere accesso a tutto perché noi abbiamo il privilegio completo su quel computer;
- **Necessità di sapere:** significa che dobbiamo consentire l'accesso a qualsiasi componente del sistema solo ai dati che sono necessari per eseguire quel compito. Consideriamo Amazon: diverse persone lavorano al suo interno. Quando si effettua un ordine su Amazon c'è una personast che prende l'ordine. Dall'ordine può vedere solo ciò che l'utente ha ordinato, ma la persona non sa chi ha ordinato e la destinazione della merce.

Banca Centrale Europea

Il 31 gennaio 2013 la BCE ha formulato alcune "*Raccomandazioni per la sicurezza dei pagamenti via Internet*". In generale, quando c'è un regolamento ci sono raccomandazioni **generiche**, ma in genere i giuristi non vogliono entrare nei dettagli. Nello scenario della sicurezza questo è cambiato di recente. Non è possibile lasciare che siano le singole aziende a decidere quali siano le regole migliori per la sicurezza. Queste raccomandazioni si applicano alle autorità di governo degli schemi di pagamento, ai fornitori di servizi di pagamento (PSP), agli esercenti che utilizzano le carte di credito per i pagamenti.

Le principali raccomandazioni sono:

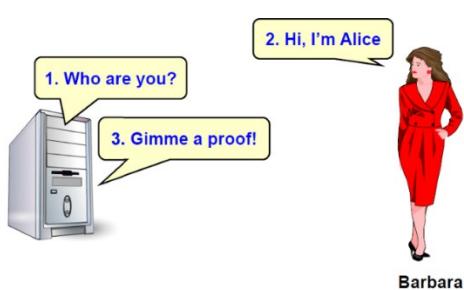
- Proteggere l'avvio dei pagamenti via Internet e l'accesso ai dati sensibili dei pagamenti con una **forte autenticazione del cliente**;

- **Limitare il numero di tentativi di accesso o di autenticazione**, definire regole per il "time out" delle sessioni dei servizi di pagamento via Internet e impostare limiti di tempo per la validità dell'autenticazione;
- Stabilire **meccanismi di monitoraggio delle transazioni** progettati per prevenire, individuare e bloccare le transazioni di pagamento fraudolente.

- **Fornire assistenza** e indicazioni ai clienti sulle migliori pratiche di sicurezza online, impostare avvisi e fornire strumenti per aiutare i clienti a monitorare le transazioni.

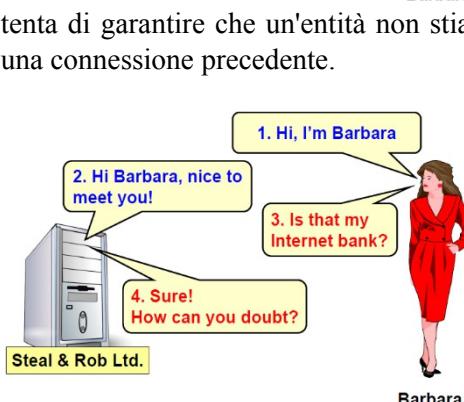
Sicurezza proprietà

- Autenticazione (semplice/mutuale);
- Autenticazione tra pari;
- Autenticazione dei dati/origine;
- Autorizzazione, controllo degli accessi;
- Integrità;
- Riservatezza, privacy, segretezza;
- Non ripudio;
- Disponibilità;
- Tracciabilità, responsabilità.



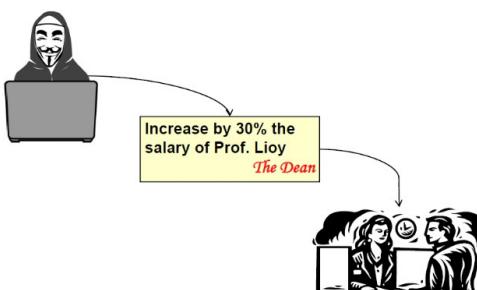
Autenticazione tra pari (semplice)

Quando c'è una comunicazione tra due peer che partecipano alla comunicazione, devono autenticarsi. Due entità sono considerate peer se implementano lo stesso protocollo in sistemi diversi; ad esempio, due moduli TCP in due sistemi comunicanti. Il sistema informatico di solito pone alcune domande come quelle in figura. In genere per una prova si fornisce una password. Questa è l'**autenticazione semplice**: solo una parte si autentica. Essa tenta di garantire che un'entità non stia eseguendo una mascheratura o una riproduzione non autorizzata di una connessione precedente.



Autenticazione reciproca tra pari

Nell'autenticazione reciproca tra pari è necessaria anche una prova formale della connessione al server reale. Un server falso potrebbe visualizzare la stessa pagina della banca normale solo per ottenere le credenziali dall'utente. Quando ci connettiamo a un server, di solito ci fidiamo del server, ma abbiamo bisogno di una prova. Entrambe le parti si autenticano a vicenda.



Autenticazione dei dati

Qualcuno potrebbe scrivere un'e-mail per aumentare lo stipendio di qualcuno, firmata dal direttore, ma non ci sono prove che l'e-mail provenga dal direttore. Un'e-mail elettronica non ha un'autenticazione adeguata. Lo stesso accade per i file. I dati normalmente non sono autenticati, quindi potrebbe essere necessario un sistema di autenticazione anche per i dati.

Non ripudio

Il non ripudio è una prova formale, accettabile da una corte di giustizia, che fornisce una prova *innegabile* del creatore dei dati. Il non ripudio impedisce al mittente o al destinatario di negare un messaggio trasmesso. Questo ha diverse implicazioni perché non abbiamo bisogno solo dell'autenticazione, ma anche dell'**integrità**, perché se qualcuno modifica i dati di un documento, questo dovrebbe essere rilevato perché non è più il documento originale autenticato. C'è una differenza tra l'autenticazione e l'identificazione:

l'autenticazione significa che abbiamo usato qualche mezzo elettronico per provare l'identità (per esempio nome utente e password), ma cosa succede se la password è stata

rubato? Siete davvero voi o qualcun altro? Al contrario, l'identificazione è molto più forte, ad esempio il Touch ID degli smartphone, perché l'utente è l'unico che può eseguire tale operazione.

Esempio di non ripudio

Consideriamo il non ripudio di una firma elettronica:

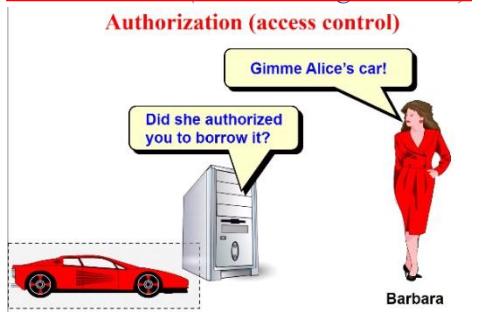
- Sintassi (è la tua firma?)
- Semantica (avete capito cosa stavate firmando?) - ciò che non capite non ha valore legale (ad esempio le piccole righe di un documento che non sono comprensibili).
- Testamento (avete firmato volontariamente?)
- Identificazione (eri davvero TU il firmatario?)
- Tempo (quando avete firmato?)
- Luogo (dove avete firmato?)

La firma elettronica è un insieme di bit che rappresentano la firma di una persona. Se non sappiamo chi deve posizionare questi bit per rappresentare la firma, non possiamo sapere di quale persona si tratta. Per questo motivo esistono delle specifiche per la firma elettronica.

Disponibilità

Un servizio di disponibilità è un servizio che protegge un sistema per garantirne la disponibilità. Questo servizio risponde ai problemi di sicurezza sollevati dagli attacchi denial-of-service. Dipende dalla gestione e dal controllo adeguati delle risorse del sistema e quindi dipende dal servizio di controllo degli accessi e da altri servizi di sicurezza.

Autorizzazione (controllo degli accessi)



Autenticazione significa identificare chi sono gli attori del sistema. Dopo l'autenticazione possiamo prendere decisioni. Ad esempio, nella figura Barbara si è identificata correttamente e poi chiede al computer "apri la scatola dell'auto di Alice" e il computer esegue una **decisione di autorizzazione (o controllo di accesso)**. Questo perché l'auto nella scatola non è l'auto di Barbara. Quindi, c'è anche un sistema che deve sapere se qualcuno è autorizzato o meno a eseguire un'operazione.

Una differenza importante!

- L'autorizzazione consiste nel verificare se l'utente, che è già stato autenticato, è autorizzato o meno a eseguire un'operazione;
- L'autenticazione è l'identificazione degli utenti di un sistema.

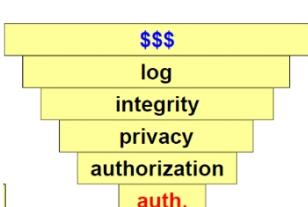
Piramide della sicurezza

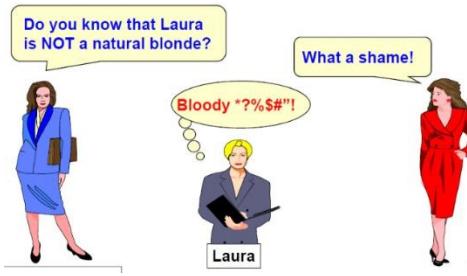


L'autenticazione è la caratteristica più basilare del nostro sistema: se non identifichiamo gli attori non possiamo salire in cima alla piramide. Sulla base dell'autenticazione ci sono: l'autenticazione (chi può fare cosa), la privacy (chi può leggere), l'integrità (chi può modificare), il log (prendere nota delle azioni).

Sfortunatamente, la maggior parte dei sistemi ha un'autenticazione molto debole basata su nome utente e password e, oltre a ciò, si costruisce un sistema molto complesso con

un valore enorme. Se le autenticazioni vengono violate, tutti gli altri meccanismi di sicurezza vengono compromessi. Spesso questo accade nelle aziende perché i dirigenti non comprendono i principi tecnici necessari per capire come devono essere messi in atto i diversi meccanismi di sicurezza.





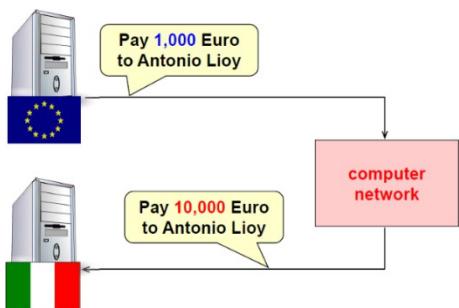
Privacy (comunicazione)

La privacy ha diversi significati: la privacy della comunicazione si riferisce, ad esempio, al fatto che quando 2 pari stanno comunicando non dovrebbe essere possibile per un terzord capire la comunicazione.

Anche se qualcuno può leggere i dati che passano attraverso la rete, nessuno (tranne i due utenti finali) dovrebbe essere in grado di capire cosa c'è dentro la comunicazione.

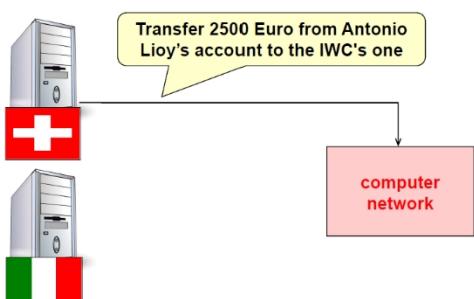
Privacy (dati, azioni, posizione)

- **Privacy dei dati:** anche se abbiamo accesso al luogo in cui sono conservati i dati, potremmo non essere in grado di leggerli;
- **Privacy delle azioni:** le aziende hanno il diritto di tracciare i siti web visitati dagli utenti, così come le forze di polizia in base alla legge antiterrorismo. Tutti i dati inviati in Italia vengono conservati per 7 anni in caso di future indagini;
- **Privacy della posizione:** questi tipi di informazioni sono disponibili per chi gestisce la rete. Nel definire le proprietà di sicurezza di un'applicazione, dobbiamo preoccuparci dei dati, delle azioni eseguite sulla rete ed eventualmente della posizione da cui è stata effettuata la comunicazione.



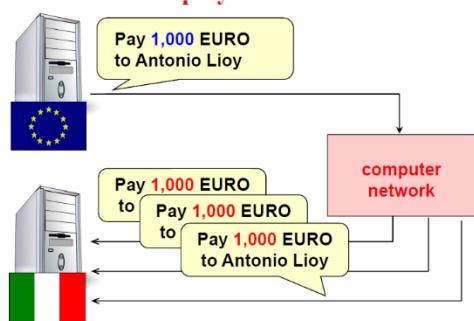
Integrità (modifica dei dati)

Significa che se i dati sono stati modificati, siamo in grado di rilevarlo. Non significa che nessuno possa modificare i dati: è impossibile. Il gestore della rete può sempre leggere ed eventualmente modificare i dati, per questo l'integrità si riferisce al rilevamento dei dati modificati.



Integrità (cancellazione/filtraggio dei dati)

I dati possono anche essere cancellati, quindi dobbiamo assicurari che se avviene la cancellazione dei dati, la rileviamo. Questo è più difficile da rilevare, perché il destinatario non riceve nulla (e non ha indizi che il pagamento nell'immagine debba essere ricevuto).



Attacco di risposta Reply Attack

I dati inviati in rete possono essere criptati e non più modificabili dal gestore della rete. Ma è possibile registrare il messaggio inviato alla rete e rispondere più volte. L'autenticazione passerà perché il messaggio non viene modificato e non è facilmente comprensibile per gli sviluppatori.

Durante la spiegazione delle proprietà della sicurezza, abbiamo sempre parlato di protezione dei dati. Esistono due tipi di protezione dei dati:

- **Dati in transito:** quando i dati vengono trasmessi su un canale di comunicazione;
- **Dati a riposo:** quando i dati sono memorizzati in un dispositivo di memoria.

Dov'è il nemico?

Per difendere qualcosa, dobbiamo sapere dove si trova il nemico. Ci sono alcune possibilità:

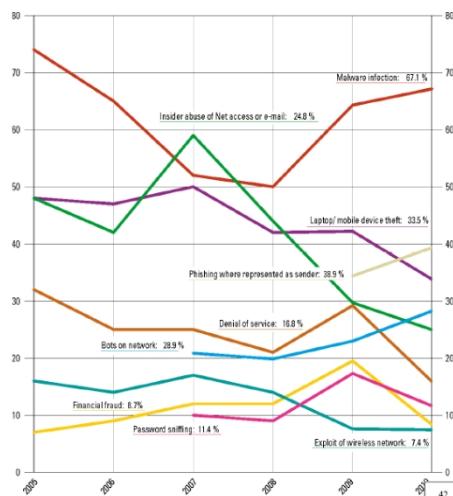
- **Al di fuori della nostra organizzazione:** In questo caso è necessaria una difesa perimetrale (o di confine) chiamata *firewall*.
- **All'esterno della nostra organizzazione, ad eccezione dei nostri partner:** in questo caso il firewall necessita di un'integrazione con alcuni percorsi o radici che siano protetti e permettano la comunicazione tra utenti fidati. In altre parole, è necessaria una VPN di protezione extranet, ovvero una intranet estesa anche ai partner.
- **All'interno della nostra organizzazione:** In questo caso dobbiamo proteggere la rete locale (LAN) e le applicazioni intranet, il che è piuttosto difficile poiché dobbiamo condividere le informazioni tra gli utenti della stessa rete.
- **Ovunque:** poiché l'attaccante può trovarsi all'interno o all'esterno dell'organizzazione, può essere ovunque, perché oggi possiamo collegarci all'organizzazione da ogni parte del mondo. **In questo caso la sicurezza deve essere posta a livello di applicazione.** In secondo luogo, poiché le applicazioni trattano dati, questi devono essere protetti indipendentemente dal luogo in cui sono memorizzati. Ad esempio, se si utilizza Dropbox, ma non è considerato sicuro dall'azienda, è possibile crittografare i dati prima di caricarli su Dropbox.

Origine dell'attacco (2016)

Le statistiche sono difficili da fare perché le aziende non rivelano di essere state attaccate, quindi le statistiche non vengono aggiornate spesso. La percentuale di attacchi esterni/interni di una statistica di Verizon è:

- Interno 20%
- 80% esterno

Quando si leggono le statistiche bisogna fare attenzione a chi sono le persone delle aziende intervistate. Verizon è uno dei maggiori provider di Internet al mondo, quindi i suoi clienti sono utenti connessi a Internet e la maggior parte degli attacchi proviene da Internet. Se le statistiche provenissero da attacchi locali, come quelli delle industrie manifatturiere, le statistiche sarebbero diverse.



Il tipo di attacco cambia nel tempo. Il grafico a sinistra mostra l'andamento degli attacchi ogni anno, ma le infezioni da malware sono le più diffuse. Questo perché, anche se esistono gli antivirus, a causa del numero di nuovi malware sviluppati ogni mese c'è sempre la possibilità che un malware sfrutti una vulnerabilità sconosciuta (vedi Finestra di esposizione).

Dal grafico si nota anche che il furto di laptop o dispositivi mobili è piuttosto frequente. Il problema del furto di laptop/smartphone non è solo la perdita economica per sostituire il dispositivo rubato, ma anche la perdita di dati che diventano indisponibili (nel caso in cui l'utente non avesse un backup) e la diffusione di informazioni riservate all'interno dei dispositivi.

Un esempio è l'articolo del Corriere del 2009 che dice: "PC americani venduti al mercato di Peshawar: Computer dell'esercito Usa con dati riservati venduti per 650 dollari lungo la strada dove le truppe Nato vengono attaccate dai talebani. Ancora pieni di informazioni riservate, come nomi, siti e punti deboli".

([corriere.it](#), 14/02/2009)

Per questo oggi le aziende pensano che i dati sui dispositivi mobili debbano essere protetti: backup e crittografia.

Insicurezza: le radici profonde

- "La tecnologia di attacco si sta sviluppando in un ambiente open-source e si sta evolvendo rapidamente": Questa nota fa un paragone poiché la maggior parte degli strumenti di attacco sono rilasciati open-source: è possibile utilizzarli in modo semplice perché non c'è bisogno di pagare, e in secondo luogo se qualcuno vuole fare un miglioramento, non deve riscrivere il sistema da zero ma può apportare piccoli miglioramenti. Il modo di attaccare migliora con piccole modifiche in modo molto veloce.
- "Le strategie difensive sono reazionarie": significa che siamo sempre in ritardo. Ci aspettiamo un nuovo attacco e, quando questo avviene, capiamo che c'è un nuovo attacco e che è necessaria una nuova protezione. Ma fino a quando ciò non accade, non viene utilizzata alcuna protezione. L'attaccante è sempre un passo avanti.
- "Migliaia - forse milioni - di sistemi con scarsa sicurezza sono connessi a Internet": Dovremmo preoccuparci della sicurezza di ogni computer connesso alla rete, non solo del nostro. Questo perché tutti possono essere infettati da malware e se ciò accade il computer non è più solo "vostro", ma è nelle mani degli aggressori. La debolezza della sicurezza di un computer è un problema per le altre persone connesse;
- "L'esplosione dell'uso di Internet sta mettendo a dura prova il nostro scarso talento tecnico. Il livello medio degli amministratori di sistema... è diminuito drasticamente negli ultimi 5 anni": con le interfacce gradevoli sui computer la produttività aumenta, ma il talento tecnico diminuisce (ad esempio, le linee di comando di Linux contro l'interfaccia di Linux con le finestre). Per essere bravi nella cybersecurity le conoscenze tecniche sono necessarie e indispensabili;
- "Un software sempre più complesso viene scritto da programmatore che non hanno ricevuto alcuna formazione sulla scrittura di codice sicuro": **codice sicuro** significa codice che funziona e non può essere facilmente sfruttato. Il **codice di sicurezza** è un codice dotato di crittografia di sicurezza, firme digitali, ecc. Il codice di sicurezza incorpora funzioni di sicurezza. Non tutti i programmatore devono conoscere le funzioni di sicurezza, ma devono essere consapevoli del codice sicuro. Ogni volta che creiamo un'interfaccia che richiede un input da parte dell'utente, dobbiamo verificare che l'input sia nella forma corretta. Accettare solo ciò che è accettabile e rifiutare tutto il resto;
- "Gli attacchi e gli strumenti di attacco trascendono i confini geografici e nazionali": Ogni volta che svolgiamo un'indagine è sempre dopo l'attacco. Ed è sempre un problema perché iniziamo a seguire la catena (qualcuno ha attaccato da un server italiano, ma il server italiano è stato attaccato da un server francese, e così via) fino ad arrivare in alcuni Paesi dove c'è l'ultimo punto in cui non c'è una legge sulla cybersecurity. E se non ci sono leggi nessuno saprà chi ha effettuato l'attacco;
- "La difficoltà delle indagini penali sulla criminalità informatica, unita alla complessità del diritto internazionale, rende improbabile il perseguimento dei reati informatici".

Problemi di base (tecnologici)

- *Le reti sono insicure:*
 - La maggior parte delle comunicazioni avviene in chiaro (a meno che non si compiano azioni);
 - Le LAN operano in broadcast (invio del messaggio a tutti e "se non è per te non leggere");
 - Le connessioni geografiche NON vengono effettuate tramite linee dedicate end-to-end, ma tramite linee condivise o router di terze parti;
- Autenticazione utente debole (normalmente basata su password);
- Non c'è autenticazione del server;
- Il software contiene molti bug.

Alcune classi di attacchi

Un mezzo utile per classificare gli attacchi alla sicurezza è rappresentato dagli attacchi passivi e dagli attacchi attivi. Un **attacco passivo** tenta di apprendere o utilizzare informazioni dal sistema, ma non influisce sulle risorse del sistema. Esempi di attacchi passivi sono:

- **Packet sniffing:** il contenuto dei pacchetti di rete (ad esempio, password e/o dati sensibili) viene

letto da terzi (non autorizzati);

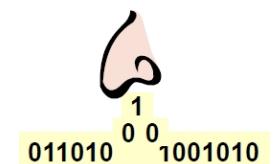
- **Analisi del traffico:** anche se il contenuto dei pacchetti non può essere compreso da una terza parte, un avversario potrebbe estrarre alcune informazioni sulla natura della comunicazione in corso.

Un **attacco attivo** tenta di alterare le risorse del sistema o di influenzarne il funzionamento, comportando una modifica del flusso di dati o la creazione di un flusso falso. Esempi di attacchi attivi sono:

- **IP spoofing/shadow server:** qualcuno utilizza l'indirizzo di un altro host per sostituirlo come client (e nascondere le proprie azioni) o come server;
- **Connection hijacking/data spoofing:** dati inseriti/modificati/annullati durante la trasmissione;
- **Denial-of-service (DoS distribuito):** la funzionalità di un servizio viene limitata o interrotta (ad esempio, ping bombing).

Sniffing dei pacchetti (intercettazione)

Lo sniffing dei pacchetti si riferisce alla lettura dei pacchetti indirizzati a un altro nodo di rete. È facile da realizzare in una rete broadcast (ad esempio, LAN) o nei nodi di commutazione (ad esempio, router, switch). Ciò è possibile mettendo la scheda di rete in modalità promiscua, il che significa che la scheda leggerà ogni pacchetto che passa attraverso il dispositivo.



Questo tipo di attacco permette di intercettare qualsiasi cosa (password, dati, ...). Alcuni possibili

Le contromisure sono: non utilizzare reti broadcast o criptare il payload del pacchetto (se non è possibile utilizzare una rete non broadcast).

Analisi del traffico

È più sottile dello sniffing dei pacchetti. Supponiamo di avere un modo per mascherare il contenuto dei messaggi o di altro traffico di informazioni in modo che gli avversari non possano capire il contenuto dei messaggi da una terza parte. Anche con la crittografia in atto, l'avversario potrebbe determinare la posizione e l'identità degli host comunicanti e potrebbe osservare la frequenza e la lunghezza dei messaggi scambiati. Queste informazioni potrebbero essere utili per indovinare la natura della comunicazione in corso.

Spoofing IP (masquerading)



Per spoofing IP si intende la falsificazione dell'indirizzo di rete di origine. In genere viene falsificato l'indirizzo di livello 3 (IP), ma è altrettanto facile falsificare l'indirizzo di livello 2 (ad esempio, ETH, TR, ...) Un nome migliore sarebbe **spoofing dell'indirizzo sorgente**. Questo è tipicamente utilizzato per gli attacchi in cui non è necessaria una risposta. Se tutti si trovano nella stessa sottorete, grazie alla funzione di broadcast, è possibile leggere anche le risposte. Attacchi di questo tipo sono: **falsificazione dei dati e accesso non autorizzato ai sistemi**. La contromisura per questo tipo di attacchi non è mai utilizzare l'autenticazione basata sugli indirizzi: in generale, gli indirizzi di rete non dovrebbero essere attendibili.

Negazione del servizio (DoS)

Il termine "denial-of-service" si riferisce al fatto di tenere occupato un host in modo che non possa fornire i propri servizi. Ad esempio, nella pubblica amministrazione per un bando di concorso, si possono inviare offerte fino a una data. Possiamo fare un'offerta e impedire a tutti gli altri di inviare e-mail. Una possibile soluzione è quella di inviare tonnellate di e-mail tenendo il server occupato fino al messaggio "Il messaggio non è stato consegnato perché la casella di posta di destinazione è piena". Abbiamo saturato il servizio di posta. Altri esempi:

- **Saturazione della posta/log** come spiegato in precedenza;
- **Ping flooding ("ping bombing"):** la richiesta di echo ICMP di solito utilizza un numero ridotto di bit (8 byte) e avvia un timer in attesa di una risposta per qualche secondo. Potremmo utilizzare il maggior numero di bit possibile: 64 Kbyte. Dovremmo inviare molte richieste di echo, alla massima velocità senza far partire i timer. In questo modo l'host sarà occupato a rispondere a tutti questi

pacchetti e non sarà in grado di eseguire altre operazioni.

- **SYN flood:** è una forma di attacco denial-of-service in cui un attaccante avvia rapidamente una connessione a un server senza finalizzarla. Il server deve spendere risorse in attesa di connessioni semiaperte, che possono consumare risorse sufficienti a rendere il sistema non rispondente al traffico legittimo (vedere TCP SYN flooding).

Di solito, gli attacchi DoS bloccano l'uso di un sistema/dispositivo e non esistono contromisure perché non è possibile sapere se chi si connette al servizio lo tiene occupato di proposito o meno. In altre parole, il DoS è come un elevato numero di clienti che utilizzano quel servizio. Il monitoraggio e il sovradimensionamento possono mitigare gli effetti. Ogni volta che viene segnalato il superamento di una certa soglia, ad esempio la saturazione delle risorse, è il momento di indagare. Potrebbe trattarsi di un problema di sistema o di sicurezza. Il responsabile della sicurezza e quello del sistema devono lavorare insieme.

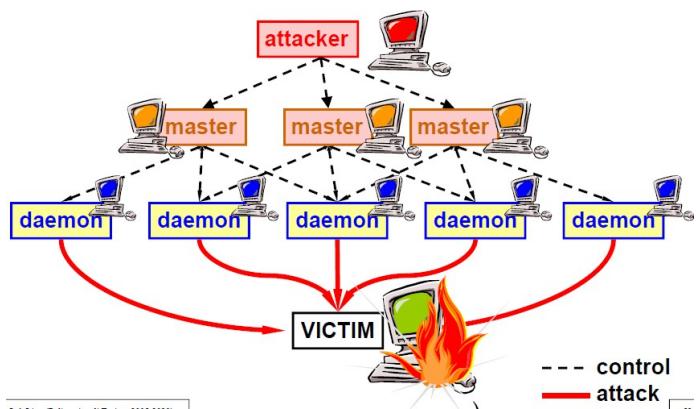
Denial-of-service distribuito (DDOS)

Il DDOS è lo stesso tipo di attacco del DOS, ma moltiplicato per il numero di aggressori che attaccano contemporaneamente. In genere, un piccolo numero di individui prende il controllo di alcuni nodi installandovi software DoS, ognuno dei quali viene spesso chiamato **demone**, **zombie** o **malbot** con lo scopo di creare una **Botnet**. Si tratta di una rete di robot (slave) controllati da un **master** che solitamente utilizza un'**infrastruttura C&C** (command & control), client-server o peer-to-peer. Inoltre, la comunicazione tra gli zombie e il master è criptata o passa attraverso canali "nascosti" (ad esempio, le informazioni passano attraverso pacchetti UDP contenuti nel payload dei messaggi ICMP Request). I canali occulti hanno lo scopo di fuorviare le indagini, infatti utilizzando UDP come payload di ICMP Request è possibile nascondere questo tipo di traffico, poiché la presenza di messaggi ICMP è comune nel traffico normale.

Questi bot sono sofisticati: in diversi casi hanno la capacità di aggiornarsi *automaticamente* al tipo di attacco più recente e, aumentando il numero di demoni, è possibile moltiplicare l'effetto dell'attacco. Altri modi per migliorare l'attacco possono essere l'uso di un "**riflettore**", sfruttando un componente di terze parti potenzialmente legittimo per inviare il traffico dell'attacco a una vittima; questa tecnica presenta due vantaggi principali:

- **nascondere l'identità degli aggressori:** gli aggressori inviano pacchetti ai server reflector con un indirizzo IP di origine impostato sull'IP della vittima (IP spoofing), quindi sommergono indirettamente la vittima con i pacchetti di risposta;
- **moltiplicare l'effetto:** può essere utilizzato anche un "**fattore di amplificazione**" N:1 (che dipende dal protocollo) in cui l'attaccante utilizza tipicamente un *server reflector* che ha una dimensione di risposta molto più grande della dimensione della richiesta. Ad esempio, se si effettua una richiesta specifica al DNS, la risposta del DNS può essere pari a 70 volte la richiesta.

Attacco DDOS



Ci sono un aggressore e una vittima. L'aggressore ha creato in qualche modo una rete di demoni, quindi seleziona alcuni nodi come master (più di uno di solito, per ridondanza), che fanno parte della botnet. L'attaccante invia l'indirizzo IP da attaccare e poi si *disconnette dalla rete* per evitare di essere rintracciato. Gli altri master coordinano il lavoro dei demoni, che devono eseguire l'attacco nello stesso momento. I master non partecipano direttamente all'attacco, cercano di rimanere nascosti il più possibile per evitare di essere uccisi a seguito di un'indagine, perché questo

interromperebbe l'attacco. L'enorme classe di fattori di amplificazione che il demone può creare fa morire la vittima. I master e i demoni non sono "persone reali", ma parte della botnet. L'unico umano è l'attaccante, che

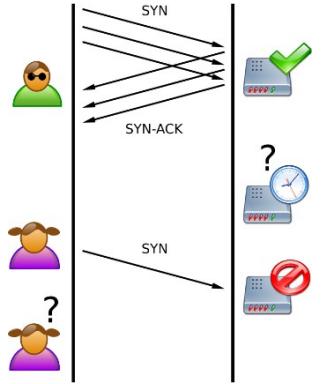


Figura 8 SYN Flood. L'aggressore (Mallory) invia diversi pacchetti ma non invia l'"ACK" al server. Le connessioni sono quindi semi-aperte e consumano le risorse del server. Alice, un utente legittimo, tenta di connettersi ma il server rifiuta di aprire la connessione, causando un denial of service.

si disconnette dopo aver iniziato l'attacco DDOS.

Caso storico: DDoS verso la server farm di Yahoo

Il primo attacco noto di questo tipo risale all'8 febbraioth, 2000 alle ore 10.30 (PST) contro la Yahoo Server Farm. Gli amministratori hanno redatto un rapporto. Il rapporto diceva:

- *"Il flusso iniziale di pacchetti, che in seguito ci siamo resi conto essere superiore a 1G bit/sec, ha messo fuori uso uno dei nostri router...".*
- *"... dopo il ripristino del router, abbiamo perso tutto il routing verso il nostro ISP a monte ..."*
questo accade perché una connessione a Internet è una connessione che richiede 2 router. I dipendenti di Yahoo hanno riavviato il proprio router, ma all'altro capo del collegamento di comunicazione il router era ancora fuori uso.
- *"... era un po' difficile capire cosa stesse succedendo, ma per lo meno abbiamo notato un sacco di traffico ICMP ...".*
- *"... alle 13.30 abbiamo ripristinato il routing di base e poi ci siamo resi conto che eravamo sotto attacco DDoS".*

In seguito l'attacco è stato ricondotto al quindicenne canadese Michael Calce (alias MafiaBoy). Anche se non è sempre possibile arrivare all'aggressore, l'avvocato statunitense può facilmente raggiungere chi sono i demoni e farsi rimborsare da loro (anche se il demone è stato vittima dell'aggressore, ad esempio se l'aggressore può prendere il controllo di un computer per scopi DDoS).

Un'interessante implicazione legale per i nodi zombie: "Siate sicuri o sarete citati in giudizio":

"È molto probabile che, se il vostro sito è stato dirottato per un attacco di tipo denial of service, possiate essere responsabili per i danni. Consiglierei sicuramente ai clienti di fare causa".

Nick Lockett, avvocato specializzato in commercio elettronico presso Sidley & Austin

Case History: DDoS verso il blog "Krebs on security"

Il 27th settembre 2016, l'amministratore del blog ha ricevuto un attacco DDoS che ha generato 665 Gbps ed è stato generato da una botnet di dispositivi IoT (o ha dichiarato di essere tale). Non sono stati utilizzati riflettori o amplificatori, ma solo milioni di dispositivi che hanno eseguito richieste perfettamente valide. Sembrava che milioni di utenti volessero connettersi al sito web nello stesso momento. Questo ha generato un traffico tale che, anche se il blog era ospitato su Akamai, il traffico era così grande che la società ha dovuto arrendersi e ha reso il blog irraggiungibile eliminando quella destinazione dalla propria tabella di routing, il 29th settembre. Non si conosce il motivo dell'attacco, forse è collegato all'analisi di Krebs di attacchi simili contro i server dei giochi online.

Server Shadow

Negli attacchi di tipo **shadow server** un host riesce a mostrarsi (alle vittime) come fornitore di servizi senza averne il diritto. Esistono due tecniche:

- **Se è in grado di sniffare la richiesta e la risposta di spoofing più velocemente del server reale,** quest'ultimo non sarà in grado di comunicare (perché il pacchetto 2nd sarà scartato, in quanto considerato un duplicato);
- **routing o manipolazione del DNS**, mappando il nome reale sull'IP del server ombra.

Gli attacchi che possono essere effettuati sono:

- **Rilasciare risposte sbagliate**, fornendo così alle vittime un servizio "sbagliato" invece di quello reale;
- **Cattura dei dati della vittima** forniti al servizio sbagliato.

La contromisura a questo tipo di attacco consiste nel richiedere l'**autenticazione del server**.

Dirottamento della connessione / Man In The Middle (MITM)

Questo tipo di attacco consiste nel prendere il controllo di un canale di comunicazione per inserire, eliminare o manipolare il traffico. Questo può avvenire in modo logico (cambiando il percorso della rete) o fisico (riuscendo a raggiungere fisicamente un router o uno switch). Gli attacchi possono essere effettuati per diversi motivi: lettura, inserimento di dati falsi e modifica dei dati scambiati tra due parti. Il

Le contromisure sono: **autenticazione, integrità** ("è lo stesso o è stato modificato?") e **serializzazione** (nessun pacchetto aggiunto o cancellato. I pacchetti arrivano nello stesso modo in cui sono stati inviati) di **ogni singolo pacchetto di rete.**

Trojan / Man In The Browser (MITB)

Si chiama così perché i canali di rete stanno diventando sempre più protetti. La maggior parte dei siti web non utilizza più *http* ma *https*: ciò significa che eseguire un attacco MITM non è più così banale. **In questo tipo di attacco, l'aggressore fa installare un software dannoso facendo affidamento sui terminali degli utenti meno protetti** (raramente gli smartphone sono dotati di antivirus, i dispositivi IoT hanno solitamente misure di sicurezza deboli, ecc. **Può trattarsi di un keylogger che regista tutto ciò che viene scritto sulla tastiera o i clic del mouse, oppure di estensioni del browser.** Questo attacco è anche chiamato "Man At The End" (MATE). La sicurezza della connessione di rete non è una contromisura per questo tipo di attacchi.

Zeus

L'attacco Zeus è noto anche come **Zbot**. Attualmente è un importante malware combinato con una botnet. È stato scoperto nel 2007 e poiché la polizia sta cercando il proprietario di questa rete, il proprietario ha dichiarato di averla venduta nel 2010. Può essere utilizzato:

- Direttamente: ad esempio, MITB per il keylogging o il form grabbing (software che legge i dati in un modulo);
- Indirettamente: per caricare altro malware (come il ransomware CryptoLocker).

È difficile da scoprire e rimuovere perché si nasconde con tecniche stealth e ci sono circa 3,6 milioni di copie attive solo negli Stati Uniti.

Bug del software

Anche il miglior software contiene bug che possono essere utilizzati per vari scopi. Il modo più semplice per sfruttare i bug del software è creare un Denial of Service. Un esempio è stato l'attacco contro il server WinNT (3.51, 4.0). Gli aggressori hanno scoperto che il server WinNT era ospitato sulla porta TCP 135 (non documentata) e hanno cercato di comunicare su tale porta inviando 10 caratteri casuali e poi CR. In questo modo il server è diventato indisponibile (100% di carico della CPU anche se non viene svolto alcun lavoro utile). Il problema era che sulla porta 135 c'era un nuovo servizio Microsoft solo che era una chiamata di procedura remota che aveva un bug: se riceveva un pacchetto malformato entrava in un loop infinito chiedendo: "dov'è un pacchetto buono?". Ma poiché l'ARP Server(?) fa parte del kernel del sistema operativo, occupava il 100% della CPU senza possibilità di essere interrotto. Microsoft ha sviluppato una soluzione con il SP3 in cui ha corretto questo bug.

Alcuni problemi tipici a livello di applicazione

- **Overflow del buffer**
avviene quando un programmatore alloca un buffer di memoria, ma non verifica che i dati memorizzati non trabocchino in quella parte di memoria. In questo modo un aggressore può tipicamente iniettare codice dannoso.
- **Memorizzare informazioni sensibili nei cookie**
in questo modo le informazioni sensibili possono essere lette da terzi (in transito o localmente sul cliente)
- **Memorizzare le password in chiaro in un DB**
In questo modo sono leggibili da terzi (ad esempio dall'operatore di backup che ha l'obbligo di fare una copia dei dati).
- **"inventare" un sistema di protezione**
Il problema sono i rischi di una protezione inadeguata

Virus & Co. (malware)

- Un **virus** è un programma dannoso che danneggia il bersaglio e poi si duplica e viene propagato dall'uomo involontariamente.
- Un **worm** danneggia il bersaglio indirettamente, solo perché si replica così tanto da raggiungere la saturazione delle risorse. Inoltre, il worm cercherà di propagarsi automaticamente. Un worm è solitamente più

difficile da individuare perché il danno creato sembra un'attività normale, a meno che non si analizzi attentamente il modello di rete;

- Il **cavalo di Troia** è un programma che trasporta del malware. Può trattarsi di un programma valido che installa anche del malware;
- La **backdoor** è un pezzo di software che fornisce un punto di accesso non autorizzato (illegale ma comune negli sviluppatori: se uno sviluppatore teme di non essere pagato può lasciare una backdoor per ottenere l'accesso in caso di problemi di pagamento);
- Un **rootkit** è un insieme di strumenti che fornisce accesso privilegiato. È nascosto (forse un programma modificato, una libreria, un driver, un modulo del kernel o un ipervisore) e non è visibile. Per questo motivo, è difficile da rilevare e rimuovere.

Virus e worm (malware)

Virus e worm richiedono una sorta di complicità (che può essere involontaria) da parte **dell'utente** (gratis, gratuito, urgente, importante), **del gestore del sistema** (configurazione errata), **del produttore** (esecuzione automatica, affidabile). Alcune contromisure sono la *consapevolezza dell'utente*, la *corretta configurazione/il software sicuro*, l'*antivirus* (installato e aggiornato).

Catena alimentare del malware

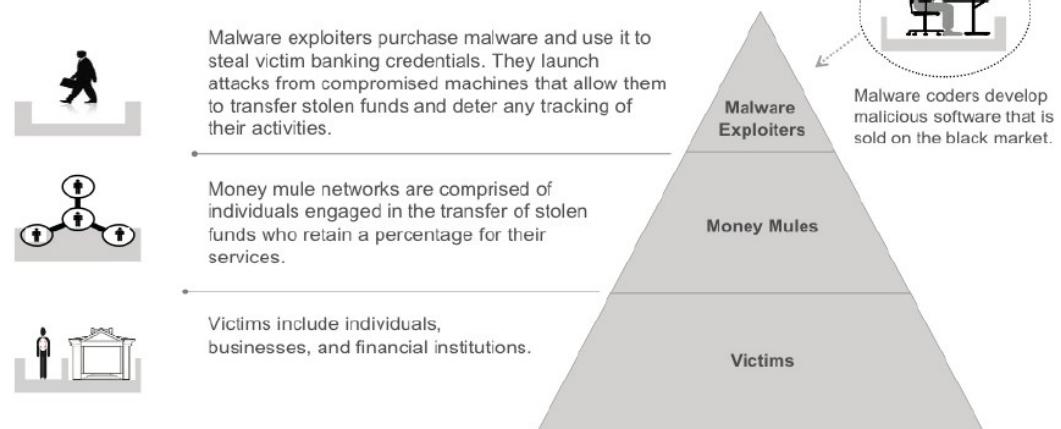
La vulnerabilità scoperta da qualcuno può essere utilizzata per sviluppare codice dannoso (ad esempio per accedere a una pagina e modificare alcuni dati). Queste informazioni possono essere sfruttate in due modi:

- Eseguire l'attacco solo per entrare in una sorta di "Hall of Fame", soprattutto in passato;
- Più recentemente, per lavoro, vendono le informazioni sulla vulnerabilità o un codice funzionante che la sfrutta sul **mercato delle vulnerabilità**, che è un mercato nascosto che si svolge solo su Internet su alcuni server fittizi che appaiono per un paio d'ore di notte.

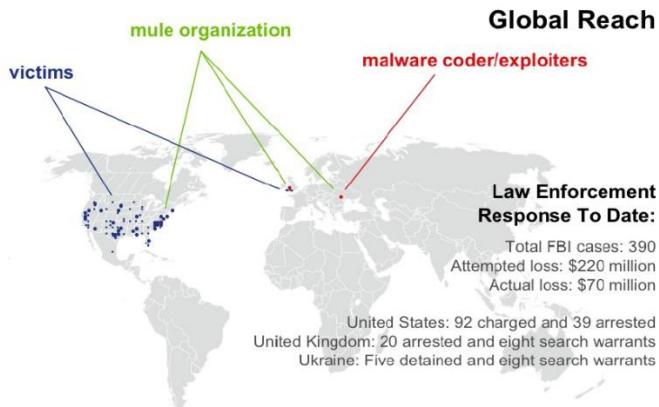
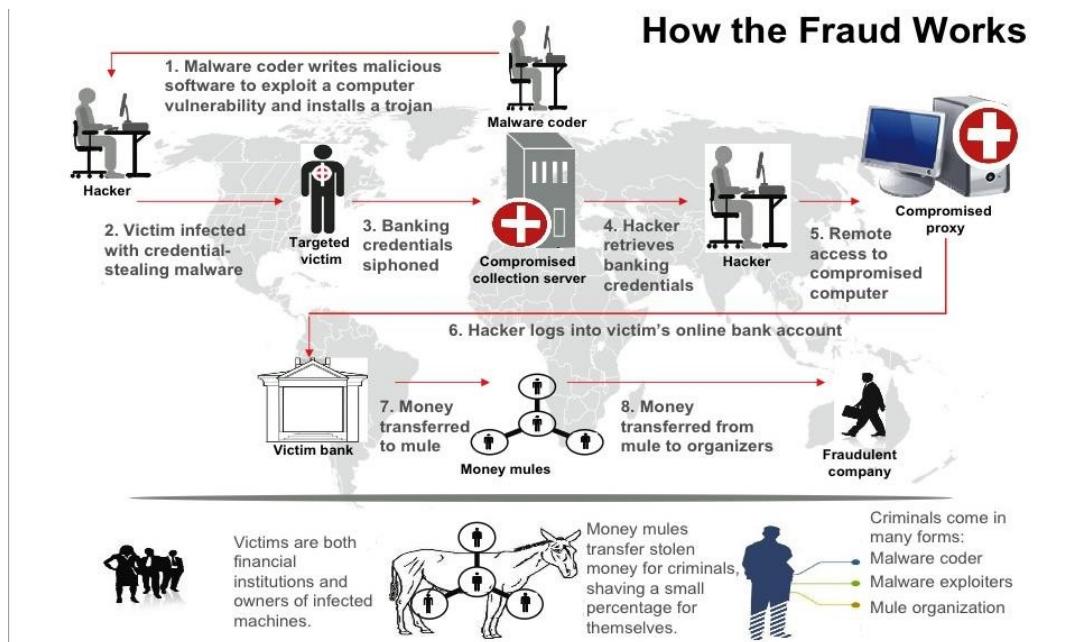
È possibile pagare con i bitcoin per comprare/vendere la vulnerabilità. Le persone che acquistano queste vulnerabilità sono i **produttori di toolkit di malware**, persone che creano programmi di attacco che necessitano di malware. Questo tipo di software viene poi utilizzato dai distributori di malware (spammer, proprietari di siti web).

Zeus

Cyber Theft Ring



Capita spesso di ricevere un'e-mail che chiede di inviare denaro al di fuori del Paese in cambio di un'ingente somma di denaro. Se accettate la richiesta, riceverete il denaro. I truffatori iniziano a fare un test: vi inviano una piccola somma di denaro (ad esempio 10.000 dollari) e vi chiedono di inviarne 7.000 in Nigeria e di tenere per voi i 3.000 dollari. Se accettate, riceverete i soldi perché stanno rubando i soldi dal conto bancario di qualcuno e li inviano a voi (così sembrate essere i responsabili) e li invierete tramite MoneyGram. Le persone in Nigeria s'aspettano, ma voi sarete i responsabili della polizia. Questo è un caso comune di **money mule** (accettare denaro da un conto bancario rubato e inviarlo a destinazione).



La maggior parte delle vittime risiede negli Stati Uniti o nel Regno Unito. L'organizzazione del mulo è diffusa in molti Paesi: Stati Uniti, Europa, Irlanda britannica, ma i codificatori/esploratori di malware sono tipicamente nell'Europa orientale. Nell'immagine sono riportati anche i numeri relativi alla risposta delle forze dell'ordine.

Ransomware

Un ransomware è un malware orientato a ottenere un **riscatto** (chiedendo denaro per rilasciare qualcosa). Se il ransomware colpisce un computer, in genere rende il contenuto del disco illeggibile (tipicamente crittografato). È valido anche per tablet e smartphone (ad esempio per cambiare la password di accesso ai dati). Non sempre, dopo aver pagato la somma di denaro richiesta (tipicamente in bitcoin), i dati possono essere recuperati: questo accade perché la chiave per sbloccare i file è memorizzata tipicamente su un server che potrebbe essere chiuso a causa di indagini della polizia. In questo modo l'aggressore non ha più la proprietà della chiave utilizzata per bloccare i dati.

Ransomware-as-a-service

Per qualche tempo è esistito un ransomware-as-a-service, in cui qualcuno forniva la fonte del ransomware per utilizzarlo anche se non ne capite nulla. Si chiamava **TOX malware (server nella rete anonima TOR)** che chiedeva il riscatto e gestiva il pagamento (con una commissione del 20% per il servizio). Il "cliente" ha il solo compito di distribuirlo alle vittime (ad esempio lasciando una penna USB in luoghi pubblici). Ha avuto una rapida crescita (1000 clienti/settimana, 100 infezioni/ora).

Anche se TOX è stato fermato dalla polizia, il ransomware esiste ancora. La tecnologia non è sufficiente per proteggersi dal ransomware. Sicuramente è utile avere installato un buon sistema anti-malware. Questo tipo di attacco non riguarda solo gli utenti finali, ma anche i server.

- **Dati criptati**

I ransomware solitamente criptano i dati che possono essere ripristinati solo con una chiave appropriata. Il **backup** può essere una possibile soluzione.

- **Quanto è vecchio il backup?**

Il backup potrebbe essere vecchio, ma supponiamo di eseguire un backup ogni giorno. Gli sviluppatori di ransomware sono a conoscenza dei backup, quindi hanno sviluppato il **ransomware silenzioso**. Il ransomware viene installato nel sistema, ma non critta il disco, bensì i backup. Continua a farlo per molti giorni (rendendo i backup illeggibili) e solo in seguito si scopre che i backup sono danneggiati;

- **Backup off-line o di rete?**

I backup devono essere eseguiti offline. Basti pensare a un caso reale di uno studio dentistico in cui si utilizzava un Network- Attached-Storage (NAS) per eseguire un backup continuo dello storage locale sullo storage di rete: un ransomware ha trovato questo tipo di configurazione e ha crittografato anche i dati sullo storage di rete, rendendo inutili i backup.

- Ma, anche se si esegue il backup offline (collegando l'hard disk esterno solo durante il backup e poi staccandolo), è possibile che un ransomware silenzioso possa vedere lo storage esterno quando è collegato e criptarlo?

Sì, è possibile. Per eseguire un buon backup è necessario utilizzare una tecnica **di backup invertita**: il disco di backup non è collegato direttamente al pc di destinazione (quello

i cui dati devono essere sottoposti a backup), ma ci dovrebbe essere un altro nodo che monta in remoto il disco del PC di destinazione, che è

visto come un dispositivo, esegue una copia e poi si scollega dal PC di destinazione. In questo modo, anche se il PC di destinazione è infetto, ciò non influisce sul processo di backup. Si noti che il nodo che esegue il backup NON deve essere collegato alla rete.

È comunque possibile che il malware abbia colpito il driver utilizzato dal PC di destinazione per esportare il disco (ad esempio il demone NFS in Linux), ma è improbabile.

- **Backup verificato o "affidabile"?**

La domanda è: avete verificato la correttezza del backup o vi siete semplicemente fidati? C'è un famoso caso reale in Svezia di backup costituiti da nastri magnetici che si sono corrotti perché durante il trasporto il sedile riscaldato ha generato corrente sui fili, generando campi magnetici che hanno danneggiato i nastri. Questo fatto è stato scoperto solo quando i dati sono stati danneggiati e si è reso necessario ricorrere ai backup.

- **Quando è avvenuto l'attacco?**

Per sapere qual è il backup corretto, è necessario sapere quando è avvenuto l'attacco. C'è stato il caso di un archivio video in cui un giornalista ha chiesto di consultarlo. Il giornalista, cercando i videoclip, ha notato che a volte il nome dei file era giusto, altre volte no. Sì, c'è un backup, ma quale backup è quello corretto? Quando è stato compiuto l'attacco? Se non si sa quando è avvenuto l'attacco, non è possibile sapere quale sia il backup corretto da ripristinare.

Problemi di base (non tecnologici)

L'immagine dice una grande verità: raffigura una corrispondenza tra tutte le tecnologie a sinistra, come firewall, antivirus (sicurezza dei dati), mentre a destra c'è l'errore umano, che è il problema. Lo scopo della tecnologia può essere vanificato da un errore umano (intenzionale o meno).

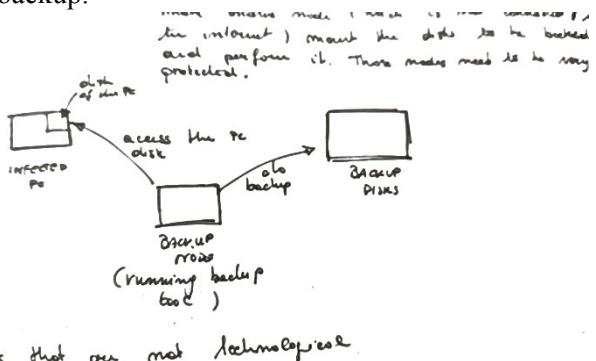


Figura 9 Uno schema che dovrebbe assomigliare al meccanismo di un backup invertito



Figura 10 Anche gli strumenti tecnologici più sofisticati possono essere resi inutili dall'errore umano.

utente e dare origine a comportamenti errati

- **Diminuzione delle prestazioni dovuta all'applicazione di misure di sicurezza**

Esempio: un'azienda ha chiesto a Lioy di fornire assistenza per l'antivirus. Questa azienda sosteneva di avere il miglior antivirus, ma era spesso infettata da virus che bloccavano il sistema. Lioy si è recato presso l'edificio dell'azienda e ha controllato tutto (antivirus installato e aggiornato). Sembrava che tutto fosse a posto. Lioy iniziò a camminare per l'ufficio per capire il problema: camminando notò un desktop con l'icona dell'antivirus disattivata. Lioy ha chiesto all'impiegato perché fosse offline, e l'impiegato ha risposto che l'antivirus controlla ogni file in lettura/scrittura e questo aumenta l'attesa per ottenere un file. In questo modo il dipendente ha trovato un modo per disattivare l'antivirus, lavorare con il computer e poi riaprirlo. Tutti i dipendenti lavorano in questo modo in ufficio. Lioy si è recato alla sede centrale e ha comunicato di avere un problema non tecnologico.

Ingegneria sociale

Nel contesto della sicurezza informatica, l'**ingegneria sociale** è la manipolazione psicologica delle persone affinché compiano azioni o divulgino informazioni riservate. Di solito il bersaglio sono gli **utenti ingenui** (ad esempio "cambia immediatamente la tua password con la seguente, perché il tuo PC è sotto attacco"), ma anche gli **utenti esperti** sono presi di mira (ad esempio copiando una mail autentica ma cambiandone l'allegato o l'URL) e avviene tramite posta, telefono o persino carta.

Esempi di ingegneria sociale

- **Phishing** (pronuncia "fishing"):

Si tratta di una tecnica di attacco basata sull'attrazione di un utente di servizi di rete verso un server fasullo (shadow server), utilizzando la posta elettronica o la messaggistica istantanea, per acquisire le sue credenziali di autenticazione o altre informazioni personali, oppure per convincerlo a installare un plugin o un'estensione che si rivela essere un virus o un trojan. Un esempio di messaggio può essere il seguente: "*Gentile utente dell'Internet banking, la preghiamo di compilare il modulo allegato e di restituircelo al più presto ai sensi della legge sulla privacy 675...*". Esistono alcune varianti:

- **Spear phishing**, quando il messaggio include diversi dati personali per mascherare il messaggio falso come un messaggio valido (ad esempio, indirizzo di posta elettronica, nome del dipartimento/ufficio, numero di telefono, ecc;)
- **Whaling**, quando l'utente bersaglio occupa una posizione di responsabilità o di potere, come un CEO o un CIO (vedi sotto). Ciò comporta due vantaggi principali dal punto di vista dell'aggressore: il danno provocato dall'inganno di questo tipo di persone può essere molto maggiore, in quanto le loro credenziali possono dare all'aggressore un grande potere; queste persone occupano generalmente una posizione commerciale e amministrativa, il che di solito

- **Bassa comprensione del problema (consapevolezza)** Le persone in generale non capiscono che esiste un problema di sicurezza. Inoltre, le persone non dispongono di misure di sicurezza e le installano solo dopo che il problema si è presentato. Questo accade perché non pensano che un problema possa accadere loro.

- **Errori degli esseri umani (soprattutto quando sono sovraccarichi e stressati):**

È molto più frequente quando siamo stressati o c'è molto lavoro.

- **Gli esseri umani hanno una tendenza naturale a fidarsi.** Di solito non sospettiamo degli altri. Poiché online non incontriamo persone, è facile avere la tendenza a fidarsi degli altri.

- **Le interfacce/architetture complesse possono trarre in inganno**

significa che non hanno le conoscenze necessarie per gestire i problemi di sicurezza;

- **Pressione psicologica:** di solito si ottiene in due modi:
 - Sfruttando l'empatia umana nei confronti di un amico o di un collega, ad esempio generando un messaggio falso che trasmette un messaggio del tipo "Aiutami, altrimenti mi troverò nei guai...", portando a fare

qualche azione che potrebbe essere contraria alle best practice, creando così un'opportunità di attacco;

- Fingendo di essere un capo, chiedendo anche qualcosa che potrebbe essere proibito, sotto la minaccia "fallo, o lo riferirò al tuo capo..." (vedi sotto).

In generale, le persone che conducono questo tipo di attacchi psicologici cercano di mostrare una conoscenza delle procedure, delle abitudini e del personale dell'azienda, per guadagnare fiducia e far abbassare le **difese** al bersaglio.

- **Pharming:** termine di uso controverso, racchiude un insieme di diverse tecniche per reindirizzare un utente verso un server ombra, tramite:

- modificare il file "hosts" del client;
- cambiare i puntatori dei server dei nomi sul client;
- modificare i nameserver di un server DHCP (ad esempio, un router ADSL/wireless);
- avvelenare la cache di un server di nomi.

Questo può avvenire tramite un **attacco diretto**, se esiste una vulnerabilità o una configurazione errata, oppure tramite un **attacco indiretto** con un virus o un worm.

Posta elettronica falsa/IM

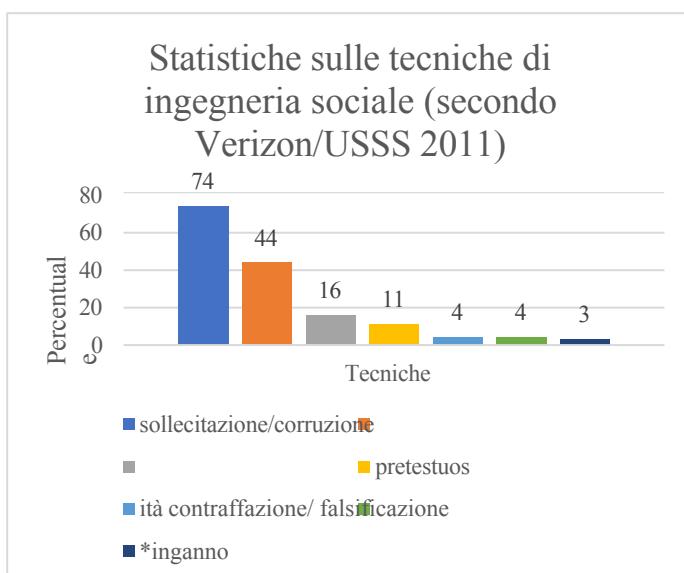
È facile creare una mail falsa, ma è difficile usare il "tono" giusto (ad esempio, *"firmo le mail come Antonio Lioy, o solo Antonio, o A. L.?"*) altrimenti la mail falsa verrà individuata. È anche possibile creare un falso SMS o IM. Ad esempio, può essere inviato:

- *Falsi messaggi di prelievo dal bancomat* che chiedono di chiamare un numero che chiederà le vostre credenziali.
- *Falso allarme rapimento*: questo accade spesso, tipicamente tramite IM. Capita spesso che molti uomini anziani si rivolgano alla polizia dicendo: "aiutatemi perché la mia ragazza è stata rapita e vogliono 200.000 euro per liberarla". Questa ragazza è una donna conosciuta sui social che crea una finta relazione con l'uomo per creare empatia e indurre la vittima a dare soldi alla finta ragazza.

Case History: *"Il signor Confindustria di Bruxelles ingannato da un hacker: 500.000 euro persi. Licenziato".* ([Giornale di Repubblica del 30 settembre 2017](#))

"Trasferite immediatamente mezzo milione su questo conto bancario estero. Ma questa mail era di un hacker. E i soldi sono spariti. Il falso ordine era (apparentemente) firmato dal direttore Panucci: "Eseguite e non chiamatemi perché sono fuori sede con il presidente". ...".

Messaggio finale: tutti i dipendenti dovrebbero essere formati sui problemi di sicurezza della vita odierna.



Tecniche di ingegneria sociale

Statistiche tratte da un vecchio sondaggio di Verizon in cui il punto importante è che quando si cerca di spiegare l'ingegneria sociale ai dipendenti, questi devono essere consapevoli di ogni tipo di attacco, come segue:

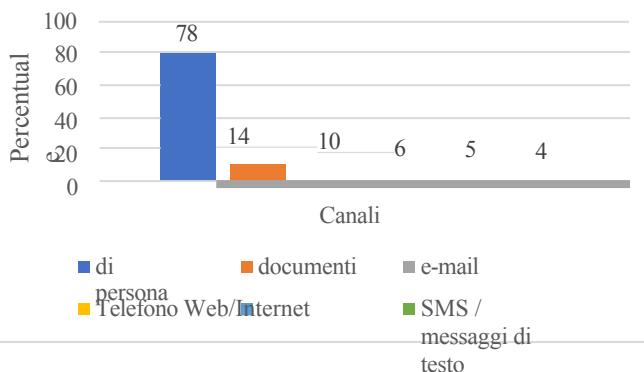
- (74%) sollecitazione/corruzione;
- (44%) di pretestuosità;
- (16%) contraffazione/falsificazione;
- (11%) *ing (cioè phishing, pharming, ...);
- (4%) bufala / truffa;
- (4%) tattiche di influenza;
- (3%) estorsione/ricatto.

Canali di ingegneria sociale

Lo stesso accade per i canali utilizzati. Potrebbe essere sorprendente ma, in questa statistica, il 78% degli attacchi di social engineering è avvenuto di persona, il che significa che qualcuno si è presentato fingendo di essere qualcun altro.

Ora prenderemo in considerazione alcuni attacchi famosi per capire cosa li ha resi possibili: quando c'è un attacco l'attenzione non deve essere rivolta all'attaccante, ma a ciò che è possibile imparare dall'attacco.

Percentuale di utilizzo del canale negli attacchi di social engineering (secondo Verizon/USSS 2011)



Storia del caso: Attacco T.J. Maxx (2007)

Nel 2007 c'è stato un attacco contro T.J. Maxx (una catena di negozi). L'attacco è avvenuto dall'esterno, dove è stato possibile per l'aggressore accedere a 45 milioni di numeri di carte di credito dei clienti. L'attacco è durato a lungo, circa 18 mesi (fino a gennaio 2007). È diventato famoso perché un gruppo di 300 banche danneggiate dall'attacco (perché dovevano rimborsare i clienti) ha intentato una class action contro T.J. Maxx, chiedendo un rimborso di 10 milioni di dollari. Questo è accaduto perché, anche se nel 2007 era noto che il protocollo wireless WEP era insicuro, T.J. Maxx ha continuato a utilizzare il WEP piuttosto che il WPA. L'aggressore non aveva bisogno di entrare nel negozio, ma era seduto nel parco con un computer portatile e un software adeguato, intercettando tutte le transazioni tra il registratore di cassa e il server di back-end. L'attacco è stato eseguito da 10 persone con l'aiuto di un ex-cracker, poi assunto dai servizi segreti statunitensi: questo non è strano, infatti quando un attaccante viene individuato, invece di andare in prigione, spesso può lavorare per il governo come consulente o fermando gli attacchi.

Considerando la normativa, in generale la legge non prescrive quali misure di sicurezza specifiche adottare, ma si limita ad avvertire di predisporre le difese secondo lo stato dell'arte. Ciò significa che un responsabile della sicurezza deve essere sempre all'avanguardia per applicare le soluzioni più recenti contro i nuovi tipi di attacchi.

Case History: Un test di phishing della US Air Force si trasforma in un problema (04/2010)

Nell'aprile 2010, durante un'esercitazione di prontezza operativa (ORE) della base aerea Andersen, situata nell'isola di Guam, è stata testata la reazione degli aviatori a un'e-mail di phishing. L'e-mail inviata dai tester di sicurezza diceva che le troupe avrebbero iniziato le riprese di "Transformers 3" a Guam e invitava gli aviatori a compilare le domande su un sito Web se volevano lavorare alle riprese. Il sito web ha poi chiesto loro informazioni sensibili, e gli aviatori avrebbero dovuto essere addestrati a non fornire quel tipo di informazioni. L'esito di questa esercitazione è diventato virale perché uno degli aviatori ha pubblicato la notizia fuori dalla base, raggiungendo il mondo civile. Quando si è diffusa la voce che l'atteso film sarebbe arrivato a Guam, i media locali hanno iniziato a chiamare la base, che ha iniziato a fare chiarezza.

"La leadership della Andersen AFB si rammarica che ci sia stata confusione nel pubblico in generale riguardo a questo tentativo di phishing", ha dichiarato Andersen in un comunicato. "Speriamo tuttavia che questo dimostri che tutti gli individui devono stare attenti al pericolo reale delle e-mail di phishing e che altri possano imparare da questa esercitazione".

Alcuni importanti attacchi recenti a

Il primo attacco discusso è **Stuxnet**, il primo attacco informatico che ha creato danni fisici. Poi **Black Energy**, il primo dei più noti attacchi utilizzati contro un'infrastruttura critica (ad esempio la distribuzione di energia elettrica). Infine, **Mirai**, **BlueBorne** e **BrickerBot**, tutti attacchi contro IoT, sistemi embedded, settore automobilistico e dispositivi domestici.

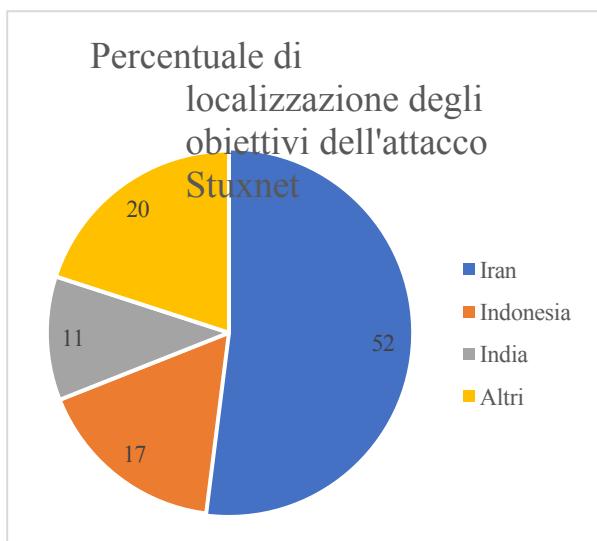
Contro i sistemi cyber-fisici

Stuxnet (2010)

È importante perché è il prototipo di un nuovo tipo di attacco, in quanto provoca danni fisici (come un virus) e tenta di propagarsi ad altri sistemi (come un worm): l'obiettivo erano i **sistemi SCADA** di uno specifico produttore collegati ai nodi infetti. SCADA è l'acronimo di Supervisory Control and Data Acquisition (controllo di supervisione e acquisizione dati) e sono quei sistemi informatici che controllano gli impianti o i macchinari di processo, ad esempio i sistemi di produzione di automobili, alimenti e così via. Poiché lo SCADA è l'interfaccia tra computer e realtà fisica, un attacco contro di esso potrebbe danneggiare i componenti del sistema fisico. Stuxnet è stato un attacco molto sofisticato perché questo worm conteneva al suo interno quattro vettori di attacco:

- uno era basato su una vulnerabilità già nota (per la quale esisteva una patch);
- una sfruttava una vulnerabilità già nota, ma per la quale non era disponibile alcuna patch (per coloro che hanno ottenuto la patch per la prima);
- due vulnerabilità "zero-day" (se esistono contromisure per la vulnerabilità nota)

Motivazioni dell'attacco



Anche la tempistica e il luogo sono particolari. Innanzitutto, il 17/6/10 è stato creato il primo danno e si è saputo che Stuxnet esisteva. Per questo motivo sono iniziate le indagini e una settimana dopo, il 24/6/10, gli analizzatori hanno rilevato l'uso di un primo certificato di firma e lo hanno revocato il 17/7/10, ma poi il malware ha iniziato a usare un secondo certificato di firma.

certificato di firma. Dopo la scoperta, ci è voluto un mese perché i bollettini di sicurezza del CERT (Computer

Emergency Response Team) e MS (Microsoft). Poi, Microsoft ha iniziato lo sviluppo di patch che sono state gradualmente rilasciate fino a ottobre '10. Questo malware ha interrotto la sua propagazione il 24/6/2012 (come scritto all'interno del codice stesso): ciò suggerisce che questo malware era mirato a una finestra temporale specifica.

La maggior parte degli attacchi è rivolta ai Paesi più sviluppati, ma questo malware ha colpito per il 52% l'Iran (la distribuzione geografica è rappresentata nel grafico seguente). Sembra che sia mirato a un'area specifica: *questo worm aveva l'obiettivo di distruggere l'impianto di arricchimento dell'uranio in Iran.*

Modalità di attacco

Sembra che la prima infezione sia avvenuta attraverso una penna USB inserita in uno dei computer che gestivano il sistema SCADA. Poi si è propagata attraverso tutti i computer collegati alla rete dell'impianto di arricchimento in Iran grazie ai dischi condivisi, al sottosistema di spooler Microsoft e ai bug dei servizi RPC. L'infezione tramite chiave USB è stata possibile perché all'interno di quella rete di computer non c'era una connessione a Internet e una chiave USB era necessaria per gli aggiornamenti dei sistemi SCADA. Sembra che il tecnico arrivato da un altro Paese per la manutenzione con la penna USB (e il buon software in essa contenuto), sia stato distratto in albergo da alcune ragazze mentre qualcuno aggiungeva alla penna USB il malware. Questo malware era camuffato da driver (fingevo di essere un driver) con una firma digitale convalidata da Microsoft e utilizzava due certificati diversi. Inoltre, la colpa non era solo di Microsoft ma anche degli sviluppatori, perché il sistema SCADA aveva un'unica password predefinita condivisa per il

database back-end.

Conseguenze dell'attacco

Il malware è stato in grado di spingere la rotazione dei cilindri (utilizzati per arricchire l'uranio) oltre il limite, senza allertare il sistema di monitoraggio utilizzato dai tecnici, che sono riusciti a fermare l'impianto quando il danno era già fatto. Questo ha ritardato l'Iran Uranium Program per diversi anni, per ripristinare il corretto funzionamento degli impianti.

Le lezioni apprese da questo attacco:

- La separazione fisica dei sistemi (air gap) non implica sicurezza: infatti, senza altre protezioni standard (antivirus, patch del sistema operativo, firewall) questi sistemi non sono sicuri. In questo caso specifico, i sistemi non hanno utilizzato alcun tipo di protezione standard, affidandosi troppo alla barriera fisica;
- I servizi attivi non necessari possono essere fonte di vulnerabilità: MS-RPC, code di stampa di rete condivise, dischi di rete condivisi non erano necessari ma erano ancora attivi. In generale, qualsiasi servizio non necessario dovrebbe essere disattivato;
- Dovrebbe essere utilizzato l'elenco di convalida del software da installare: sembra che il tecnico abbia copiato tutto il contenuto della penna USB, copiando anche il malware aggiunto. Avrebbe dovuto copiare solo il file necessario all'aggiornamento dei sistemi SCADA.

I fratelli di Stuxnet

I fratelli di Stuxnet si chiamano così perché hanno la stessa piattaforma di sviluppo di Stuxnet. La piattaforma di sviluppo è chiamata "*tilded*" perché eseguendo il reverse engineering del malware si è scoperto che tutte le variabili sono denominate "tilde1", "tilde2" e così via...

Duqu è stato scoperto nel settembre 2011 e non è un worm né un virus (non provoca alcun danno e non si propaga da solo). Una volta installato, Duqu raccoglie e invia informazioni sul sistema per la preparazione dell'attacco (ricognizione e intelligence, che significa capire il tipo di sistema in cui è installato e comprendere quali sono le caratteristiche del sistema, come il software installato).

Flame è stato scoperto nel maggio 2012 ed è uno *spyware di sistema* che registra il traffico di rete, l'audio, il video e la tastiera (silenziosamente). Anche in questo caso, non presenta danni fisici e si diffonde tramite USB o rete. Lo scopo di uno spyware è quello di rimanere all'interno di un sistema il più a lungo possibile per poterlo spiare. Ha anche una *backdoor* per rendere possibile la configurazione remota e l'aggiornamento dello spyware. Flame è stato attivo per due anni prima del suo riconoscimento.

Sauron - un malware per governare tutti

Sauron è stato divulgato nell'agosto 2016, ma era attivo dal 2011. Sostiene di essere stato creato dal gruppo Strider (gruppo anonimo di aggressori). Ha un **malware remsec**, che è una backdoor furtiva e un logger. È stato molto selettivo per i suoi obiettivi (alcuni individui in Russia, una compagnia aerea in Cina, un'organizzazione in Svezia e un'ambasciata in Belgio): solo 36 infezioni dal 2011. Il malware Remsec è in realtà un "**loader**", mentre l'attacco è scritto utilizzando il **modulo LUA**, un nuovo linguaggio interpretato. È sufficiente scrivere il codice in LUA e poi il loader lo esegue indipendentemente dal tipo di sistema. I moduli LUA sono: net loader (caricamento dalla rete), host loader (caricamento da penna USB), keylogger (registrazione dell'uso della tastiera), net listener (registrazione dei pacchetti inviati/ricevuti), pipe di base/avanzata (mettere qualcosa dopo un altro prima di inviare qualcosa alla scheda di rete, la pipe è nel mezzo per alterare o leggere i dati) e HTTP back-door.

Normalmente viene installato in un computer collegato alla rete, ma è molto flessibile perché può anche raccogliere dati da computer collegati in rete, esportarli in una chiave USB e inviarli a Internet mentre la penna viene utilizzata su un computer collegato. La differenza di dimensioni del contenuto della chiave USB può essere nascosta copiando i dati e contrassegnando la posizione corrispondente all'interno del dispositivo come unità di memoria "interrotta" (ad esempio, pagina), in modo che il sistema operativo non tenti di leggere o scrivere tali posizioni, ma l'aggressore sa comunque che il loro contenuto è costituito dalle informazioni rubate alla vittima.

Contro le infrastrutture critiche di

Energia nera

L'energia nera appartiene alla categoria delle "**minacce persistenti avanzate**" (**Advanced Persistent Threat, APT**), che consiste nel fatto che il sistema è compromesso con alcune tecniche sofisticate e viene continuamente monitorato da un sistema di comando e controllo esterno, in genere da persone molto

motivate (cioè la minaccia) per rimanere inosservato il più a lungo possibile (ecco perché si chiama "persistente").

La Black Energy è la terza fase evolutiva (altre evoluzioni sono sconosciute): Attacco DDoS nel 2007, attacco al sistema di controllo industriale nel 2010 e poi SCADA nel 2014. Si propaga attraverso il ransomware KillDisk.

Ora BE è alla versione 4. Black Energy è diventato famoso e ha preso questo nome perché è stato usato per attaccare la rete energetica ucraina. Per 6 ore l'elettricità di 230.000 clienti ucraini è stata interrotta.

Fondamentalmente, Black Energy era un *Trojan* che si diffondeva con e-mail di spear-phishing o documenti Microsoft Office dannosi (estensioni VBE). BE è costituito da diversi componenti con funzionalità personalizzate dagli autori del reato sulle vittime:

- Accesso e modifica dei file;
- Bypassare i meccanismi di controllo degli accessi e raccogliere le credenziali;
- Raccogliere dati sul sistema compromesso;
- Utilizzare il sistema compromesso per la scansione delle vulnerabilità di rete;
- Spiare le vittime, dagli screenshot al microfono;
- Esegue le applicazioni e avvia i servizi.

Contro Internet of Things, automotive, home

Mirai

Mirai è il più famoso **cyberworm** IOT scoperto intorno a settembre-novembre 2016. Questo tipo di attacco non danneggia direttamente, ma semplicemente satura le risorse (come il DoS). Ogni sistema infettato da Mirai diventa parte di un'enorme botnet per un DDoS su larga scala. È stato utilizzato per attacchi dirompenti come:

- [Krebs on Security](#) → fino a 620 Gbit/s
- Ars Technica → fino a 1Tbit/s
- Provider Dyn DNS → richieste da decine di milioni di IP

Le botnet sono distribuite in milioni di dispositivi IoT come telecamere, router residenziali o baby monitor, perché in questi dispositivi non è installata quasi nessuna protezione. In questo modo, Mirai può facilmente diffondersi nelle reti domestiche. Le reti veloci (come 4G e 5G) rendono il problema ancora più critico, perché questi dispositivi possono inviare molti dati al secondo. Mirai è un malware molto complesso perché non ha quasi dipendenze esterne (ciò significa che il malware è compilato con librerie statiche) ed è compilato in modo incrociato per essere eseguito su diverse piattaforme. Mirai è stato rilasciato come worm open-source: ciò significa che non solo è possibile studiarlo, ma anche che nuove varianti possono essere facilmente sviluppate da altri attaccanti. Mirai contiene una fase di scansione di propagazione per compromettere ulteriori obiettivi (inizialmente contatta altri nodi se riesce a infettarli). Inoltre, osserva il sistema vittima prima di contattare il C&C: quando Mirai viene installato su un dispositivo, non sa se si tratta di un dispositivo vero o falso (magari usato per studiare il comportamento di Mirai) e per questo motivo, quando si avvia, si connette a un C&C falso e se la comunicazione funziona significa che c'è un problema (perché la connessione non dovrebbe funzionare, il che significa che qualcuno sta dando a Mirai pieno accesso) e si spegne automaticamente. Solo dopo aver eseguito questo controllo, contatta il C&C reale. Il tipo di attacco, essendo basato sulla rete DoS, è guidato dal C&C che può decidere quale attacco effettuare e può aprire molte connessioni TCP o inviare tonnellate di pacchetti UDP o utilizzare GRE o il semplice SYN flooding.

Il codice di Mirai è stato divulgato settimane prima del primo attacco DDoS per rendere impossibile il riconoscimento degli autori. Da allora, esistono molti cloni, ad esempio:

- 900.000 router di Deutsche Telekom sono stati compromessi. Si trattava di router prodotti con Arcadian, che presentavano un noto bug TR-064 e che DT non ha corretto: così Mirai è stato facilmente installato. Mirai può attaccare anche altre vulnerabilità dei router, come il protocollo TalkTalk (se abilitato).

BashLite

Il primo famoso *malware* per l'IoT, che prende di mira principalmente telecamere/DVR. In questo caso questi elementi avevano un sistema Linux incorporato e l'attacco sfruttava il bug Shellshock nella shell Bash. Alcuni attacchi hanno avuto un'efficacia di ordini di grandezza inferiore a Mirai (alcuni attacchi hanno raggiunto il picco di 400 Mbit/s) e contattano immediatamente C&C per poi propagarsi.

BlueBorne

BlueBorne utilizza le connessioni Bluetooth per controllare i dispositivi bersaglio: sfrutta diverse vulnerabilità note, ad esempio l'overflow del buffer nella gestione della memoria dello stack di rete di Linux/Android. Inoltre, aggira i controlli di sicurezza solitamente collocati al confine "previsto" e colpisce i sistemi con intercettazione in aria, i dispositivi IoT e automobilistici: solitamente gli sviluppatori di sistemi automobilistici collocano le difese, ma solo al "confine previsto", non considerando che il Bluetooth potrebbe essere un canale di attacco.

Come in generale per gli attacchi, se non si riescono a riconoscere tutti i canali di attacco a un sistema, si rischia di proteggerne alcuni e di dimenticarne altri. BlueBorne si diffonde anche attraverso l'aria, il che significa una diffusione più rapida. Può eseguire il Man in the Middle o prendere il pieno controllo della vittima.

Esempio Immuni: ora che tutti hanno installato l'app che richiede il Bluetooth, le persone potrebbero cercare di accedere al dispositivo attraverso il Bluetooth.

Ciò richiede la progettazione di nuove protezioni, che non sono facili da realizzare. Alcune aziende sostengono di avere un nuovo programma che fornisce un'attenuazione per bloccare l'esecuzione di codice indesiderato (rilevato e iniettato tramite Bluetooth).

BrickerBot

Il caso Wind-Infostrada (abbonati alla fibra)



Bonny F.

un'ora fa



Hi guys.. Sorry for not speaking Italian. The Wind modems had telnetd running on port 8023 and a default password admin/admin which gave anyone root access to them. Unfortunately the modems got bricked by malware known as 'BrickerBot' which wrote random data over the partitions. When Wind eventually asks customers to return the devices for a replacement you'll want to be first in line..

Un messaggio è apparso sui blog e sulla chat (messaggio sul lato sinistro). In questo caso Wind ha dovuto sostituire fisicamente tutti i modem perché non erano in grado di risolvere il problema da remoto. È successo a ottobre 2017. I modem sono andati fuori uso, il che significa che è stato necessario sostituirli. L'autore di questo attacco

ha affermato di averlo fatto a fin di bene, forse per evitare la creazione di una botnet. Questo caso è simile, ma peggiore, a quello di Deutsche Telekom del 2016.

Lezioni apprese:

- **Utilizzate password forti;**
- **Permettere l'accesso amministrativo esterno solo da specifiche reti "fideate" (e non da qualsiasi indirizzo su Internet):** usare gli indirizzi IP come "filtro" per consentire l'accesso è comunque debole, ma almeno è meglio che consentirlo da tutti;
- **Applicare tempestivamente tutte le patch di sicurezza per ridurre al minimo il WOE:** nel caso Wind-Infostrada, il bug era noto, quindi l'azienda avrebbe dovuto installare le patch.

Mappe di attacco in tempo reale

Esistono molti siti web in grado di mostrare il numero di attacchi in corso. Questo è possibile perché molti di questi siti sono fornitori di soluzioni di sicurezza. Ottengono le informazioni dal proprio software, che è installato su milioni di dispositivi e rileva eventuali attacchi. I siti web hanno molte etichette che indicano esattamente di che attacco si tratta:

- **OAS = Scansione all'accesso**
Gli antimalware eseguono una scansione per rilevare l'eventuale presenza di malware prima di accedere a un file.
- **ODS = Scansione su richiesta**

- Eseguire la scansione della penna USB inserita prima di copiare i file.
- **MAV** = *Mail Anti-Virus*
Antivirus sulle e-mail che controlla gli allegati.

- **WAV** = *Antivirus Web*
Prima di scaricare qualcosa da Internet, l'oggetto viene temporaneamente immagazzinato in una struttura centrale, controllato alla ricerca di eventuali malware e solo se non c'è nulla viene trasmesso a destinazione.
- **IDS** = *Sistema di rilevamento delle intrusioni*
Monitora le reti e gli host per individuare eventuali attacchi.
- **VUL** = *scansione della vulnerabilità*
Più proattivo: alcune macchine cercano periodicamente di contattare le altre in rete e di verificare la presenza di una vulnerabilità: se viene trovato qualcosa, viene inviato un avviso.
- **KAS** = *Kaspersky Anti-Spam*
Specificamente per Kaspersky
- **BAD** = *Rilevamento attività botnet*
Rilevata attività di botnet.

Catena di morte informatica (intrusione): attacco

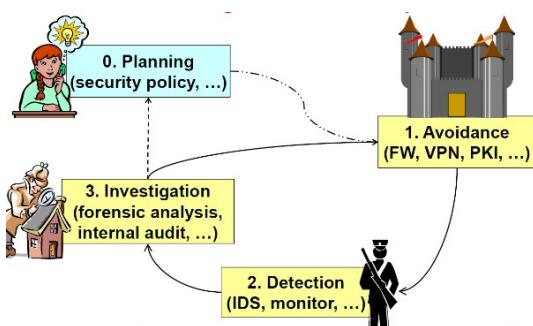
In base all'approccio adottato, in alcuni casi la seguente tabella può essere utile per descrivere un attacco.



Questa è una possibile sequenza di fasi di un attacco. Qualcuno dice che non è perfetta perché, ad esempio, la fase 2 può includere la fase di "sviluppo" (ad esempio, acquisto di domini utili, strumenti, server). È stata criticata anche perché questa descrizione si basa sul "modello difensivo perimetrale", ma che dire degli insider? Potrebbe non funzionare. Ci sono anche diverse fasi che vengono eseguite esternamente all'obiettivo e quindi sono difficili da identificare (cyber intelligence!). Esistono contromisure per quasi tutte le fasi:

- **Rilevare**: determinare se un attaccante sta curiosando;
- **Rifiuta**: impedisce la divulgazione di informazioni e l'accesso non autorizzato;
- **Interruzione**: interrompere o modificare il traffico in uscita (verso l'attaccante);
- **Degradatare**: contrattaccare il comando e il controllo (inviando dati a quel C&C);
- **Ingannare**: interferire con il comando e il controllo (modificare il traffico verso il C&C);
- **Contenere**: modifiche alla segmentazione della rete (isolare l'intrusione).

I tre pilastri della sicurezza

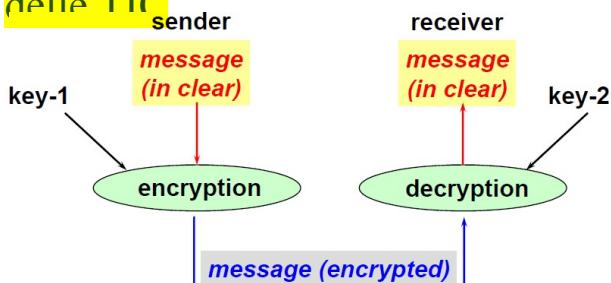


- **Pianificazione:** definire una politica di sicurezza da implementare;
- **Evitazione:** difese come firewall, VPN, PKI

Se le difese vengono superate:

- **Rilevamento:** sistema di rilevamento delle intrusioni o monitoraggio;
 - **Indagine:** una volta attaccati, eseguire l'analisi forense, l'audit interno (capire cosa è andato storto);
- L'ultima fase deve essere eseguita **prima di** regolare il sistema.

Nozioni di base sulla sicurezza delle TIC



La tecnica più utilizzata per ottenere la protezione da molti secoli è la **crittografia**: una tecnica matematica che esegue trasformazioni costituite da algoritmi di **crittografia e decrittografia**:

- l'algoritmo di crittografia prende un messaggio (in chiaro) e lo trasforma in modo che non sia più comprensibile;
- per recuperare il testo originale, l'algoritmo inverso di

La decrittazione lo renderà nuovamente leggibile.

Oltre agli algoritmi è necessaria anche una **chiave**, la chiave-1 per la crittografia e la chiave-2 per la decrittografia, che sono solo un flusso di bit.

La crittografia viene utilizzata nelle comunicazioni e per memorizzare i dati (ad esempio per memorizzare i dati su dischi senza il permesso di leggerli se non per noi). La terminologia comunemente utilizzata nella crittografia comprende altre due parole chiave:

- **Plaintext o cleartext:** il messaggio in chiaro, di solito viene indicato con **P**;
- **Cifrario:** i messaggi criptati, normalmente indicati con **C**. In alcuni paesi "criptato" suona offensivo per motivi religiosi (culto dei morti); in questi casi si preferisce "cifrato".

La forza della crittografia (principio di Kerchoffs)

Il Principio di Kerckhoffs (1883) afferma che la sicurezza di un sistema crittografico deve risiedere esclusivamente nella scelta delle chiavi; tutto il resto (compreso l'algoritmo stesso) deve essere considerato di dominio pubblico. Tuttavia, questo principio si basa sul fatto che le chiavi abbiano le seguenti proprietà:

- Sono tenuti segreti;
- Sono gestiti solo da sistemi affidabili;
- Sono di lunghezza adeguata.

Non solo non è importante che gli algoritmi di crittografia e decrittografia siano tenuti segreti, ma è meglio rendere pubblici gli algoritmi in modo che possano essere ampiamente analizzati e che vengano identificati i loro possibili difetti e vulnerabilità.

Sicurezza attraverso l'oscurità (STO)

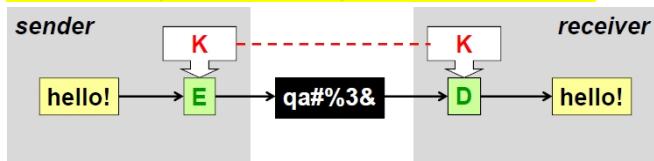
Il Principio di Kerckhoffs è legato al **concepto di Sicurezza attraverso l'oscurità**: significa che un sistema è protetto ma i dettagli su come è stato protetto non vengono divulgati. In generale, questo non è considerato un meccanismo di sicurezza valido, perché se qualcuno scopre come il sistema è stato protetto (e abbiamo visto che esistono anche modi non tecnici per farlo), non è più sicuro. Per questo motivo,

diciamo che "La sicurezza attraverso l'oscurità è una cosa tanto negativa per i sistemi informatici quanto lo è per le donne": "Gli uomini cercano di nascondere le cose alle donne, ma quando scoprono la verità, è peggio che se l'avessero scoperta fin dall'inizio" [Cit. Lioy].

Tuttavia, esiste una categoria di persone (come i militari) che tende ad applicare l'ordinanza solare, ma come livello aggiuntivo. È possibile utilizzare l'ordinanza solare come strato aggiuntivo solo se si utilizza un algoritmo veramente forte (ma non segreto).

A seconda della relazione esistente tra la chiave-1 e la chiave-2 esistono diversi tipi di crittografia.

Chiave segreta / crittografia simmetrica



Ha questo nome perché viene utilizzata una sola chiave condivisa da mittente e destinatario. Nell'immagine c'è un testo in chiaro che viene utilizzato come input per il blocco E (crittografia) inserendo anche la chiave. Il risultato è un testo comprensibile che viene inviato al destinatario.

Per recuperare il testo si utilizza l'algoritmo a blocchi D (decrittazione) con la **stessa** chiave usata per criptare il testo originale. Se si utilizza una chiave diversa, è disponibile un risultato, ma sarà sbagliato (e tipicamente comprensibile). Il problema nella figura è la linea tratteggiata: come condividere in modo sicuro la chiave tra mittente e destinatario? Le formule utilizzate sono le seguenti:

$$C = enc(K, P) \text{ o } C = \{P\} K$$

$$P = dec(K, C) = enc^{-1}(K, C)$$

La chiave è normalmente in prima posizione perché permette di utilizzare P o C avendo la chiave in evidenza. Questo tipo di crittografia ha un basso carico computazionale e viene utilizzata per la crittografia dei dati.

Algoritmi simmetrici

<i>name</i>	<i>key</i>	<i>block</i>	<i>note</i>
DES	56 bit	64 bit	obsolete
3-DES	112 bit	64 bit	56-112 bit strength
3-DES	168 bit	64 bit	112 bit strength
IDEA	128 bit	64 bit	
RC2	8-1024 bit	64 bit	usually K=64 bit
RC4	variable	stream	secret
RC5	0-2048 bit	1-256 bit	optimal when B=2W
AES	128-256 bit	128 bit	most common

Esistono molti algoritmi e la tabella a sinistra ne rappresenta solo una piccola selezione. La prima colonna indica il nome, la seconda la lunghezza della chiave e la terza l'unità di base che ogni algoritmo può crittografare: gli **algoritmi a blocchi** richiedono un'unità minima di dati. Dall'elenco, solo RC4 è un **algoritmo di flusso** che non richiede un blocco fisso, ma funziona su qualsiasi lunghezza. È anche un algoritmo segreto e non esistono informazioni su di esso (nessuna specifica o prova formale che sia veramente sicuro). L'algoritmo **DES** (per molti anni lo standard) è ora considerato obsoleto e non dovrebbe mai essere utilizzato. L'algoritmo più comune/utilizzato di questo tipo è **AES**, attualmente il più forte. **RC5** è ottimale quando la dimensione del blocco è il doppio della parola della CPU.

Figura 11 La tabella mostra due informazioni importanti su alcuni algoritmi noti: la lunghezza della chiave utilizzata e l'unità di base dei dati che l'algoritmo è in grado di gestire. Si veda in basso che esistono metodi utilizzati quando la dimensione dei dati non è pari a un blocco.

architettura (ad esempio, arco a 64 bit -> blocco a 128 bit) su cui è implementato l'algoritmo.

Perché esistono così tanti algoritmi? Perché abbiamo molti tipi di computer e molti algoritmi non sono adatti a una CPU di bassa potenza.

La funzione EX-OR (XOR)

\oplus	0	1
0	0	1
1	1	0

È l'operatore di "confusione" ideale. La particolarità di questa tabella di verità è che ha il

50% di 0 e il 50% di 1. Se lo XOR viene eseguito con 2 ingressi casuali (probabilità 0:1 = 50%:50%) anche l'uscita sarà ugualmente casuale. (ad esempio AND ha più probabilità di 0). Lo XOR non cambia la distribuzione di probabilità dell'ingresso, anche se genera uscite diverse.

Gli algoritmi a blocchi, noti anche come cifrari a blocchi, sono una classe di algoritmi crittografici utilizzati per crittografare dati suddividendoli in blocchi fissi di dimensioni specifiche e applicando una trasformazione crittografica a ciascun blocco. Questi algoritmi sono ampiamente utilizzati per proteggere la confidenzialità e l'integrità dei dati durante la loro trasmissione o archiviazione.

Ecco alcune caratteristiche chiave degli algoritmi a blocchi:

- Dimensione del Blocco: Gli algoritmi a blocchi operano su blocchi di dati di dimensioni fisse. La dimensione di un blocco varia a seconda dell'algoritmo specifico, ma tipicamente è di 64, 128 o 256 bit.
- Modalità di Operazione: Gli algoritmi a blocchi possono essere utilizzati in diverse modalità di operazione per adattarsi a diverse esigenze di crittografia. Alcune delle modalità comuni includono Electronic Codebook (ECB), Cipher Block Chaining (CBC), Cipher Feedback (CFB) e altre.
- Chiave Segreta: Per crittografare e decrittografare i dati con successo, è necessario utilizzare una chiave segreta condivisa tra mittente e destinatario. La sicurezza del sistema dipende dalla robustezza della chiave utilizzata.
- Iterazioni: Gli algoritmi a blocchi spesso operano su un numero fisso di iterazioni, o "rounds", in cui ciascuna iterazione applica una serie di trasformazioni al blocco di dati. Il numero di iterazioni varia a seconda dell'algoritmo e della sua sicurezza.
- Risultato Casuale: Lo stesso blocco di dati crittografato con la stessa chiave restituirà sempre lo stesso risultato. Tuttavia, piccole modifiche nei dati di input o nella chiave causeranno risultati completamente diversi.

Alcuni esempi di algoritmi a blocchi ben noti includono:

- AES (Advanced Encryption Standard): Un algoritmo a blocchi ampiamente utilizzato, noto per la sua efficienza e sicurezza.
- DES (Data Encryption Standard): Un algoritmo a blocchi pioniere, ora considerato obsoleto a causa della sua lunghezza chiave corta.
- Triple DES: Un'evoluzione di DES che offre una maggiore sicurezza attraverso l'applicazione di DES per tre volte con diverse chiavi.

La scelta dell'algoritmo appropriato dipende dalle specifiche esigenze di sicurezza e di prestazioni dell'applicazione.

DES

È l'acronimo di "Data Encryption Standard" ed è uno standard FIPS 46/2 (Federal Information Processing Standard, l'ente che standardizza le soluzioni per il governo americano). Le modalità di applicazione del DES a dati non equamente suddivisi in blocchi sono indicate nello standard FIPS 81. Il DES è un algoritmo strano perché ha una chiave a 64 bit, ma la sua efficacia è solo quella di una chiave a 56 bit, poiché 8 bit sono la parità degli altri. Ciò significa che quando si costruisce una chiave per l'algoritmo DES si creano solo 56 bit e ogni 7 bit l'algoritmo inserisce un bit che è la parità dei 7 bit precedenti. Quando un attaccante deve attaccare il DES, deve scoprire solo 56 bit. Questo è l'unico algoritmo che presenta una differenza tra i bit effettivi (bit utilizzati per creare la chiave) e i bit totali (quantità totale di bit). Il DES utilizza un blocco di dati a 64 bit progettato negli anni '60, quando i computer non erano così potenti. Per eseguire tutti i calcoli matematici, è stato necessario creare un'unità speciale chiamata processore di crittografia perché il DES utilizza:

- **XOR**: che non è un problema → operazione elementare;
- **Spostamento**: nessun problema → operazione elementare;
- **Permutazione**: operazione costosa. La permutazione non è casuale, ce ne sono diverse, ma comunque l'implementazione era molto più efficiente se fatta direttamente in hardware.

Triplo DES (3DES, TDES)

È l'applicazione ripetuta del DES (tre volte). È possibile utilizzare due o tre chiavi diverse a 56 bit. Di solito si applica prendendo l'input, criptandolo e ripetendo il processo altre due volte usando come input il risultato dell'operazione di crittografia precedente (in modo da applicare tre crittografie di seguito). Normalmente, viene implementato in modalità EDE, che nella sua forma standard richiede due chiavi: il testo in chiaro viene cifrato con la chiave 1, poi sull'output viene applicato l'algoritmo di decifrazione con la chiave 2 (che in realtà non decifra ma applica comunque una trasformazione) e infine quest'ultimo output viene nuovamente cifrato utilizzando la chiave 1. Questa modalità è stata scelta perché mettendo in ingresso il testo in chiaro, il testo in chiaro viene cifrato con la chiave 2 e il testo in uscita viene cifrato con la chiave 2. Questa modalità è stata scelta perché mettendo $K_1 = K_2 = K_3$ con l'EDE viene implementato anche il semplice DES. Quindi l'EEE funziona, ma EDE funziona altrettanto bene. È stato dimostrato che 3DES con due chiavi può avere garanzie di sicurezza inferiori rispetto a atteso, in particolare, nominando K_{eq} la chiave equivalente ottenuta applicando le trasformazioni discusse:

- **3DES con 2 chiavi** ha $K_{eq} = 56 \text{ bit}$ if $2^{59} B$ di memoria disponibile per l'attaccante, altrimenti $K_{eq} = 112 \text{ bit}$. Le trasformazioni applicate sono riassunte da:
$$C' = \text{enc}(K_1, P) \quad C'' = \text{dec}(K_2, C') \quad C = \text{enc}(K_1, C'')$$
- **3DES con 3 chiavi** assicura $K_{eq} = 112 \text{ bit}$:
$$C' = \text{enc}(K_1, P) \quad C'' = \text{dec}(K_2, C') \quad C = \text{enc}(K_3, C'')$$

$$\begin{aligned} C &= \text{enc}(K_1, P) & C' &= \text{dec}(K_2, C) & C'' &= \text{enc}(K_1, C'') \\ C' &= \text{dec}(K_1, C'') & C &= \text{enc}(K_2, C') & P &= \text{dec}(K_1, C) \end{aligned}$$

Il 3DES è uno standard FIPS 46/3 e ANSI X9.52 (la famiglia X9 è uno standard per la sicurezza nelle applicazioni bancarie e finanziarie).

Perché non il doppio DES?

La doppia applicazione di qualsiasi algoritmo di crittografia è soggetta a un attacco known-plaintext denominato meet-in-the-middle che consente di decifrare i dati con al massimo 2^{N+1} tentativi $N+1$ (se le chiavi sono lunghe N-bit). Se l'attaccante conosce un testo in chiaro, può sferrare l'attacco. Per questo motivo, la versione doppia degli algoritmi di crittografia non viene mai utilizzata, perché il tempo di calcolo raddoppia ma la lunghezza effettiva della chiave aumenta di un solo bit.

Inoltre, è stato dimostrato che, se l'algoritmo simmetrico di base è un gruppo, esiste una chiave equivalente K_3 tale che:

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

Ciò significa che in questo caso il tempo necessario per la normale cifratura/decifratura raddoppia, ma non si

guadagna nemmeno un bit in termini di Keq .

Attacco Meet-in-the-middle

Per ipotesi si utilizzano N chiavi di bit e si **conoscono P e C** tali che $C = enc(K_2, enc(K_1, P))$. Si noti che $\exists M$ tale che $M = enc(K_1, P)$ e $C = enc(K_2, M)$. L'attaccante calcola 2^N valori $X_i = enc(K_i, P)$ e poi calcola 2^N valori $Y_j = dec(K_j, C)$ e poi si cercano i valori K_i e K_j tali che $X_i = Y_j$. Possono esserci dei "falsi positivi", ma possono essere facilmente scartati se è disponibile più di una coppia (P, C) .

Supponiamo che un'azienda protegga le comunicazioni tra Torino e Milano con un doppio DES. Se l'attaccante sniffa la rete (e legge i dati criptati) può inviare un messaggio su quella linea (il che significa che controlla il testo in chiaro) e poi vedrà il testo cifrato (del testo in chiaro inviato) generato automaticamente dal sistema di sicurezza. Questo è il modo per ottenere una coppia di (P, C) senza conoscere le chiavi.

IDEA

Mentre il DES era gratuito, l'IDEA (**International Data Encryption Algorithm**) è brevettato con basse royalty (solo per uso commerciale, ASCOM AG). Utilizza una **chiave di 128 bit** e un **blocco di dati di 64 bit**. È piuttosto famoso perché è stato inizialmente utilizzato in [PGP](#). È adatto a essere implementato in modo efficiente via software perché utilizza tre operazioni di base: **XOR, addizione modulo 16 e moltiplicazione modulo $2^{16} + 1$** . **Osservando** queste operazioni si evince che l'idea è stata presa in considerazione per una classe specifica di CPU, quelle con parola a 16 bit, e questo suggerisce che IDEA è stato sviluppato per CPU a bassa potenza, quelle impiegate in molti luoghi. Un'applicazione di IDEA è stata fatta molti anni fa nelle competizioni di Formula 1, da quando è emerso che la McLaren spiava la Ferrari intercettando le comunicazioni radio. La Ferrari chiese ad ASCOM AG di creare dispositivi crittografati e utilizzò la crittografia IDEA perché era adatta alle radio grazie ai bassi requisiti di consumo della CPU e della batteria.

RC2, RC4

Questi due tipi di crittografia sono stati realizzati da [Ron Rivest](#) e RC sta per **Ron's Code**. Sono stati realizzati appositamente per la sua azienda, quindi sono segreti e di proprietà di RSA (l'azienda di Ron). Non è brevettato perché non viene divulgato. Questi algoritmi sono da 3 a 10 volte più veloci del DES. L'RC2 è un algoritmo a blocchi, mentre l'RC4 è un algoritmo a flusso, ed entrambi utilizzano chiavi di lunghezza variabile.

RC2 è stato infine pubblicato come RFC-2268 nel marzo 1998 e si è scoperto che utilizza chiavi da 8 a 1024 bit (di solito a 64 bit) e blocchi di dati a 64 bit. Al contrario, RC4 rimane oggi un algoritmo segreto. Poiché viene venduto sotto forma di libreria, qualcuno ha fatto reverse engineering per realizzare una possibile versione C dell'algoritmo, chiamata **ARCFOUR**.

Applicazione degli algoritmi a blocchi

"Come si applica un algoritmo a blocchi a una quantità di dati diversa dalla dimensione del

blocco dell'algoritmo?". I casi sono due:

1. Dimensione dei dati da crittografare > dimensione del blocco dell'algoritmo
 - o **ECB** (Electronic Code Book): oggi considerato non sicuro, non deve mai essere utilizzato;
 - o **CBC** (Cipher Block Chaining): attualmente è il modo migliore per applicare un algoritmo di crittografia a dati più grandi della dimensione del blocco dell'algoritmo;
2. Dimensione dei dati da criptare < dimensione dei blocchi dell'algoritmo
 - o **Padding**: da utilizzare solo quando la dimensione dei dati non è esattamente un multiplo della dimensione di un blocco, ad esempio quando i dati sono lunghi 2,6 blocchi. Si noti che questa tecnica non viene utilizzata per imbottire i dati che sono appena più corti della dimensione di un blocco;
 - **CTS** (CipherText Stealing): permette di utilizzare algoritmi a blocchi senza padding, quindi senza aumentare la dimensione del testo cifrato rispetto al testo in chiaro;
 - o **CFB** (Cipher FeedBack), **OFB** (Output Feedback);
 - o **CTR** (modalità Contatore).

ATTENZIONE! Questi NON sono algoritmi, ma modalità di applicazione di un algoritmo.

È necessario specificare: l'algoritmo, la dimensione della chiave e, se si tratta di un algoritmo a blocchi, anche la modalità dell'applicazione (ad esempio, AES-128-CBC).

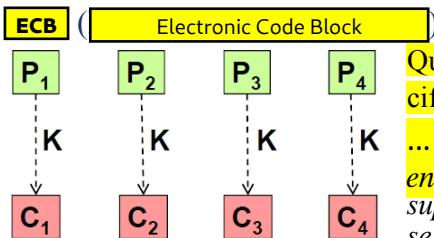


Figura 12 Modalità BCE applicata a un algoritmo: per ogni blocco $i = 0, \dots, N$,

$C_i = enc(K, P_i)$. È molto veloce poiché il calcolo del testo cifrato può essere eseguito in parallelo, ma lo stesso motivo è fonte di insicurezza

Questa modalità divide il testo in chiaro in blocchi e ogni blocco viene cifrato separatamente con la chiave; ciò significa che, per ogni blocco $i = 0, \dots, N$, $C_i = enc(K, P_i)$. Non deve essere usato! [L'io ha detto che l'esame non sarà superato se l'ECB viene utilizzato in qualche modo]. In particolare, non deve essere utilizzato per i messaggi lunghi (qualsiasi messaggio più lungo di 1 blocco) perché:

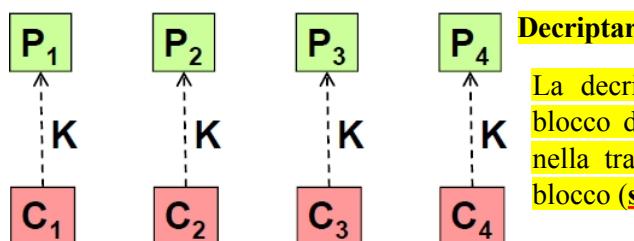
- Se l'aggressore intercetta il testo cifrato e scambia la posizione di 2 blocchi, non viene rilevato (**block swapping**). Ciò consente di scambiare dati all'interno di un messaggio crittografato;
- **Blocchi identici generano cifrature identiche**, quindi è vulnerabile agli attacchi **known-plaintext**. Gli attacchi al testo in chiaro richiedono la precompilazione di tutte le possibili cifrature di un testo in chiaro noto: confrontando

il blocco cifrato con tutte le possibili cifrature precalcolate, è possibile capire qual è la chiave.

Esempio di attacco Known-Plaintext

Supponiamo di voler intercettare un messaggio inviato dal rettore del Politecnico di Torino. È possibile ipotizzare che i suoi messaggi contengano la parola "Torino", quindi iniziamo a crittografare questa parola con tutte le chiavi possibili. Successivamente, possiamo andare a sniffare i blocchi inviati dal rettore: se uno di questi blocchi è uguale alla cifratura di "Torino", allora possiamo usare la chiave usata per ottenere quella cifratura di "Torino" per decifrare il resto del messaggio. Alcune argomentazioni contro questo metodo potrebbero essere: in che modo è scritto "Torino" (ad esempio, "TORINO", "Torino")? E se "Torino", essendo più breve di una parola macchina, fosse diviso in diversi blocchi (ad esempio, $P_1 = "...To"$, $P_{i+1} = "rino..."$)? L'attacco known-plaintext è reso possibile dal fatto che di solito le persone si scambiano dati in un *formato strutturato*, in modo che ci siano metadati che di solito sono noti (ad esempio, l'intestazione fissa).

Se un file Word viene creato con una sola parola, si tratta di grandi KB di memoria (e non di semplici byte). Questo accade perché Word applica un'intestazione al documento che è sempre la stessa (e sempre all'inizio di un file). Se il primo blocco di un file Word è criptato, può fungere da dizionario per i file Word. Ciò significa che non è necessario indovinare il testo in chiaro, ma solo il formato del file.



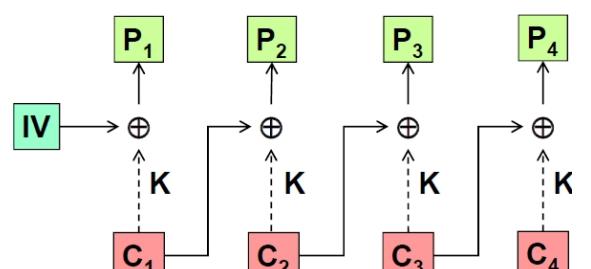
Decrittare

La decrittazione è semplice perché è necessario ottenere il blocco del testo cifrato e decrittarlo con la chiave. Un errore nella trasmissione genera un errore nella decrittazione di un blocco (**senza propagazione** dell'errore su altri blocchi).

Figura 13 Modalità BCE applicata ad un algoritmo: la decifrazione è semplice, per il blocco i_{th} è $P_i = {}^{-1}enc(K, C_i)$.

CBC (Cipher Block Chaining)

Questa modalità risolve tutti i problemi di ECB. Divide il testo in chiaro in blocchi, ma ogni blocco, prima di essere crittografato, viene sottoposto a XOR con il testo cifrato



del blocco precedente. Questo rende la crittografia imprevedibile. Nel primo blocco (che è l'intestazione del file di cui si è parlato prima) c'è un problema. Per applicare la CBC, è necessario aggiungere un elemento chiamato

IV (Vettore di Inizializzazione) che viene che ha considerato come C_0

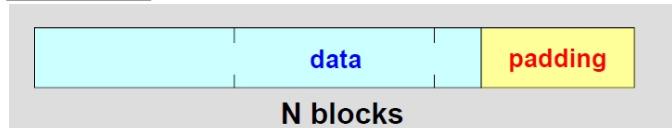
Figura 14 Modalità CBC applicata a un algoritmo: per ogni blocco $i = 0, \dots, N$, $C_i = \text{enc}(K, P_i \oplus C_{i-1})$, con $C_0 = IV$.

l'unico scopo di alterare il primo blocco di testo in chiaro. Esiste anche una protezione contro lo scambio di blocchi, perché se alcuni blocchi vengono scambiati l'output sarà diverso, in quanto saranno alterati con l'elemento sbagliato dello XOR.

Decodifica

Il vettore IV deve essere noto anche al destinatario, perché quando si riceve il testo cifrato, per eseguire la decifrazione con la chiave è necessario conoscere l'IV per decifrare il primo blocco (poiché XOR è l'operatore inverso di se stesso). L'IV deve anche essere diverso ogni volta (possibilmente casuale, per evitare di essere indovinato) perché il suo scopo è quello di rendere impossibile precompilare le possibili cifrature del primo blocco. Deve inoltre essere un **NONCE** (Numero usato una volta), il che significa che l'IV deve essere generato e mai riutilizzato in futuro. In alcuni casi l'IV viene inviato in chiaro perché, anche se l'attaccante lo conoscesse, potrebbe iniziare il calcolo solo in quel momento, e questo è un compito lungo; inoltre, gli permetterebbe di attaccare solo il messaggio inviato, perché il successivo avrà un altro IV. L'IV può anche essere inviato criptato utilizzando l'ECB, poiché si tratta solo di un blocco.

PADDING (allineamento, riempimento)



Non è sempre possibile dividere il testo in chiaro in blocchi che abbiano tutti la stessa dimensione. Nell'immagine i dati sono costituiti da due blocchi e da un pezzo che non è un blocco completo (una piccola parte alla fine). A

La tecnica di codifica della parte piccola è chiamata **padding**. Alcuni bit vengono aggiunti alla fine dei dati originali fino a raggiungere il multiplo del blocco.

Da allora ha avuto alcuni problemi:

- Il testo cifrato diventa più lungo del testo in chiaro: aggiungere dati inutili significa trasmettere/memorizzare più dati del necessario;
- Quale dovrebbe essere il valore dei bit di padding? Come possiamo distinguere il padding dal testo in chiaro originale al ricevitore?

Tecniche di imbottitura

- Se la lunghezza è nota o può essere ottenuta, come nel caso di una stringa C, è possibile aggiungere **byte nulli**, alla fine **0x00 0x00 0x00**;
- Il DES originale specificava di aggiungere un bit "1" seguito da tanti "0" quanti necessari **1000000**;
- Un byte con valore 128 seguito da byte nulli **0x80 0x00 0x00**
- Valore dell'ultimo byte pari alla lunghezza del padding **0x? ? 0x? ? 0x03** (3 byte di padding. L'ultimo ha valore 3 e il precedente ha valore da specificare).

Esistono molti tipi di tecniche, perché la tecnica di padding scelta permette di effettuare o evitare diversi attacchi.

Ci sono molti tipi di valori che possono essere selezionati (*non ricordate tutte le tecniche, ricordate solo che ce ne sono molte e che deve essere selezionata quella giusta*):

- **(Schneier) byte nulli** e. g. 0x00 0x00 0x03
- **(SSL/TLS) byte con valore L**, ad esempio 0x03 0x03 0x03
- **(SSH2) byte casuali**, ad esempio 0x05 0xF2 0x03
- **(IPsec/ESP) numero progressivo**, ad esempio 0x01 0x02 0x03
- **Byte con valore L-1**, ad esempio 0x02 0x02 0x02

Alcune note

- B sta per la dimensione del blocco dell'algoritmo
- D indica la dimensione dei dati da elaborare

Alcune di queste tecniche offrono un controllo (minimo) dell'integrità: se la chiave è sbagliata o i dati sono manipolati, i byte di padding sono incoerenti (ad esempio, lunghezza maggiore di un blocco o valori di padding errati).

In genere, viene applicata a dati di grandi dimensioni, sull'ultimo frammento risultante dalla divisione in blocchi (ad esempio, per ECB o CBC). Se $|D| < |B|$ si preferisce una tecnica ad hoc (CFB, OFB, CTR, ...). Se c'è solo un byte non è bene aggiungere 65 byte di padding.

Anche se il testo in chiaro è un multiplo esatto del blocco, è necessario aggiungere comunque un padding per evitare errori nell'interpretazione dell'ultimo blocco. Il padding più grande è richiesto quando non c'è padding da aggiungere.

Con il padding di SSH2, dati uguali danno cifrature diverse (anche se crittografate con la stessa chiave).

Il tipo di padding per un determinato algoritmo determina il tipo di (alcuni) attacchi possibili, ma dipende anche dall'algoritmo utilizzato.

Ciphertext Stealing (CTS)

Quando si utilizza il padding, la dimensione del testo cifrato è maggiore di quella del testo in chiaro. Questo potrebbe non essere accettabile in molti casi (ad esempio, dati crittografati su disco rigido che non possono essere inseriti nello stesso disco). È possibile utilizzare il CTS al posto del padding. L'ultimo blocco (parziale) viene riempito con byte presi dal penultimo blocco (crittografato), quindi questi byte vengono rimossi dal penultimo blocco (che diventa parziale). Dopo la crittografia, la posizione dell'ultimo e del penultimo blocco viene scambiata.

È utile quando non è possibile aumentare la dimensione dei dati dopo la crittografia, ma il tempo di calcolo aumenta leggermente.

Esempio di CTS con ECB (crittografia)

- 1) Il testo in chiaro viene suddiviso in blocchi;
- 2) Il penultimo blocco è pieno, quindi è possibile crittografarlo con la chiave;
- 3) L'ultimo non è pieno e crea un problema;
- 4) Il blocco cifrato C_{n-1} è diviso in due parti: **testa** e **coda**. La coda ha la stessa dimensione come parte mancante nell'ultimo blocco;
- 5) La **coda** viene aggiunta nel blocco parziale, che crea un blocco completo;
- 6) Viene trasmesso l'intero blocco creato e poi la **testa** viene trasmessa.

Il ricevitore deve invertire la procedura. I bit corrispondenti alla coda vengono crittografati due volte.

Lo scambio dei blocchi è necessario perché se si invia la testa senza la coda, l'inizio del blocco successivo viene interpretato come parte di quello precedente.

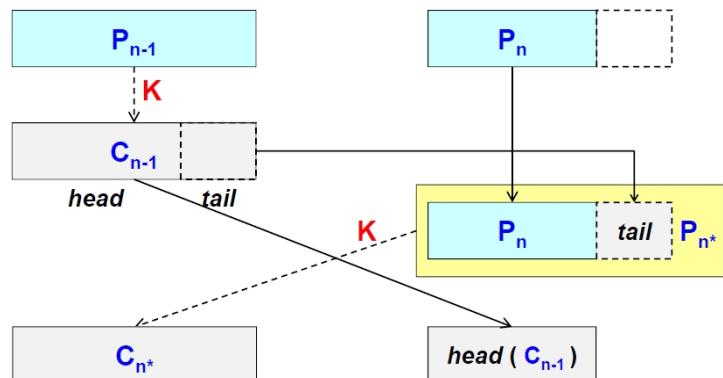


Figura 15 Esempio di CTS con ECB (crittografia): si noti tuttavia che **ECB NON DEVE MAI ESSERE UTILIZZATO**; l'esempio ha lo scopo di facilitare la comprensione del CTS.

Esempio di CTS con CBC (crittografia)

I vari blocchi vengono cifrati con la chiave e lo XOR, in particolare $CN-2$ che viene utilizzato nello XOR di P_{N-1} per la cifratura. Nell'ultimo passo

P_N è riempito di tutti gli zeri ed è XORed con la cifratura del blocco $N - 1$ e in questo modo viene cifrato l'ultimo blocco. Il blocco EN viene collocato nella posizione $N - 1$. Al contrario, la crittografia $EN-1$ non viene trasmessa completamente, ma solo la testa.

Sembra che una parte di $EN-1$ sia mancante, ma la coda viene XORed è composta da tutti zeri.

Poiché lo XOR con lo zero non altera i dati, la coda viene incorporata in EN come $N -$

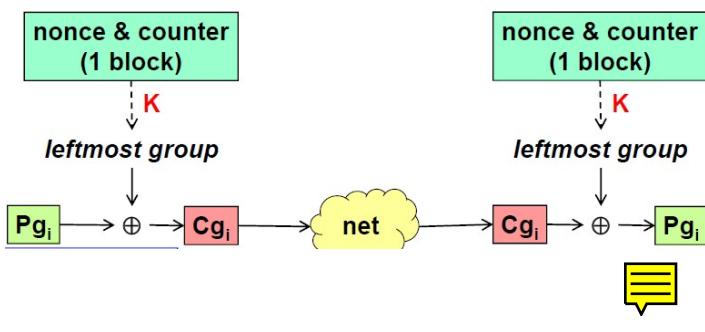
1 blocco. In questo modo il testo cifrato ha la stessa dimensione del testo in chiaro. Con questa modalità, se i blocchi sono

Il testo in chiaro viene riempito completamente dal testo in chiaro, non è necessario aggiungere alcun padding. Oggi è molto utilizzata, soprattutto per la crittografia della memoria sui dispositivi incorporati (ad esempio, gli smartphone).

CTR (counter mode)

È utile per i testi in chiaro di dimensioni inferiori a un blocco e non ha senso crittografarli utilizzando il padding. La CTR è la tecnica più utilizzata che utilizza un algoritmo a blocchi per cifrare N bit alla volta (un "gruppo", spesso un byte). Ha una buona proprietà che consente l'accesso diretto casuale a qualsiasi gruppo di testo cifrato.

Considerando la CBC, non è possibile criptare o decriptare qualsiasi blocco (perché è necessaria la concatenazione), ma qui è possibile decriptare un singolo blocco, ma richiede un nonce (chiamato anche IV) e un contatore, che vengono in qualche modo combinati (concatenati, sommati, XOR, ecc.).



Nell'esempio c'è un gruppo (ad esempio 1 byte) più piccolo di un blocco chiamato Pg_i e in cima c'è un registro grande esattamente come un blocco. Il registro viene riempito con nonce e contatore, composti nel modo che abbiamo deciso. Poiché si tratta di un blocco unico, possiamo crittografarlo direttamente con la chiave. Quindi viene preso il gruppo più a sinistra.

Ad esempio, se Pg_i è di 1 byte, solo l'ultimo byte verrà preso dalla cifratura del registro. Quindi si applicherà XOR che genererà il corrispondente testo cifrato Cg_i che potrà essere inviato attraverso la rete.

A destra, il destinatario deve avere un altro registro inizializzato nello stesso modo del mittente e fare la stessa operazione inversa. Naturalmente, mittente e destinatario devono avere lo stesso nonce e lo stesso valore nel contatore (devono essere sincronizzati). Questo tipo di trasmissione è sensibile agli **attacchi di cancellazione**, perché se un messaggio viene rimosso, non ci sarà più sincronizzazione. Questo può essere evitato con tecniche di integrità (discusse più avanti).

In sintesi, i vantaggi di questa tecnica sono:

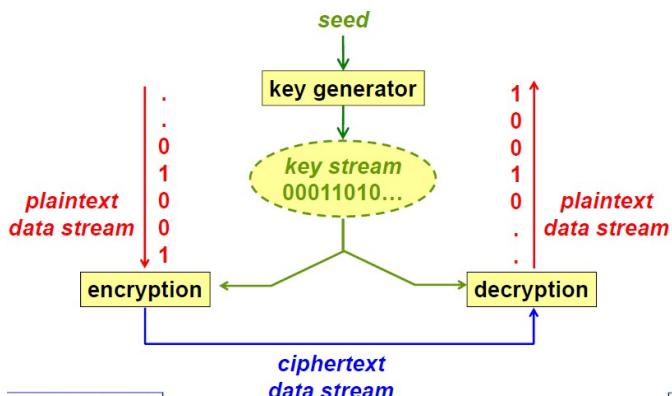
- **Accesso diretto a qualsiasi gruppo:** è possibile decriptare un gruppo a piacimento, senza prima decriptare qualsiasi altro;
- **nessuna propagazione** dell'errore su altri gruppi: se si verifica un errore durante la trasmissione, questo influisce solo sul gruppo di testo in chiaro corrispondente;

Solitamente applicata quando è necessario criptare pochi dati, la modalità contatore può essere utilizzata per criptare dati di grandi dimensioni criptando alcuni byte per volta.

Algoritmi di tipo stream

Gli algoritmi di flusso non richiedono la divisione del testo in chiaro in blocchi. In genere lavorano con 1 bit o byte alla volta. In questa categoria esiste un solo algoritmo perfetto, chiamato **one-time pad**, che richiede una chiave lunga quanto il messaggio da proteggere: per questo motivo, non ha un uso pratico.

Gli algoritmi reali utilizzano generatori di chiavi pseudocasuali, che devono essere sincronizzati tra il mittente e il destinatario (ad esempio RC4 e SEAL).



stesso seme e lo stesso algoritmo per la generazione della chiave.

Il flusso di dati in chiaro a sinistra viene crittografato un bit alla volta. Poiché il flusso di dati può essere lungo, anche il flusso di chiavi deve essere lungo. Il flusso di chiavi viene generato da un generatore di chiavi, inizializzato con un **seme** (il seme è in realtà la chiave, poiché deve essere lo stesso per il destinatario e il mittente). La funzione di crittografia sarà fondamentalmente un operatore XOR (non è necessaria una funzione più complessa) perché le operazioni saranno imprevedibili. Al ricevitore, la decifrazione sarà applicata utilizzando lo stesso flusso di chiavi, il che significa utilizzare il metodo

Se l'aggressore riesce a cancellare una parte del flusso di dati del testo cifrato, la decrittazione risulterà errata poiché utilizzerà una parte sbagliata del flusso di chiavi. D'altra parte, questi algoritmi sono molto veloci.

Curiosità

L'importanza degli algoritmi di stream è evidenziata dal concorso internazionale **ESTREAM**, che ha chiesto ai crittografi di sottoporre alla valutazione i loro migliori cifrari di stream. È stato completato nell'aprile 2008 ma rivisto periodicamente (l'ultima revisione risale al gennaio 2012). È stato selezionato un portafoglio di algoritmi (a seconda che si voglia implementare su hardware o software):

- (software, chiave a 128 bit) → *HC-128, Rabbit, Salsa20/12, SOSEMANUK*
- (hardware, chiave a 80 bit) → *Grano v1, MICKEY 2.0, Trivium*

Versioni estese:

- (software, 256 bit) *HC-256, Salsa20/12*
- (hardware, 128 bit) *MICKEY-128 2.0*

Salsa20 e ChaCha20

Si tratta di algoritmi di flusso simmetrici inventati da D. J. Bernstein con chiavi a 128 o 256 bit. ChaCha20 è un miglioramento di Salsa20 che prevede una maggiore "diffusione" dei bit (il che significa che se si cambia 1 bit dell'ingresso, vengono cambiati più bit dell'uscita) ed è più veloce su alcune architetture.

L'operazione di base è un ARX (add - rotate - XOR) a 32 bit, mentre la funzione di base è:

$$f(\text{Key256}, \text{Nonce64}, \text{Counter64}) = 512 \text{ bit keystream block}$$

Questo permette di decifrare $O(1)$ qualsiasi blocco a caso. Salsa20 esegue 20 round di miscelazione dell'input, mentre Salsa20/12 e Salsa20/8 hanno un numero ridotto di round di miscelazione: 12 e 8 rispettivamente, più veloce ma meno sicuro.

ChaCha20 è stato standardizzato ed è fortemente adottato, con documentazione RFC-8439 (Chacha20 e Poly1305 per i protocolli IETF). L'IETF ha leggermente modificato la specifica originale: nonce più grande e contatore di blocchi più piccolo. In particolare,

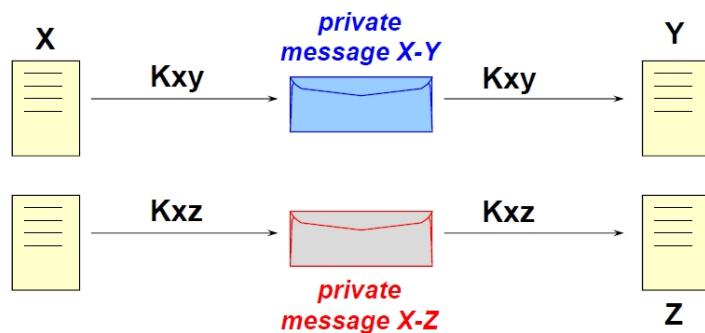
- Nonce a 96 bit;

- Contatore di blocchi a 32 bit;
- Quindi un limite di 256 GB (2^{32} blocchi da 64 byte);
- Per superare questo limite, si può utilizzare la definizione originale di Bernstein.

È diventato famoso perché Google lo utilizza (per Chrome su Android), openssh e vari [CSPRNG](#) (ad esempio /dev/urandom dal kernel Linux 4.8).

Crittografia simmetrica

Questo tipo di crittografia utilizza una chiave singola e segreta. È necessaria una chiave per ogni coppia (o gruppo) di utenti. Nell'esempio X ha una chiave condivisa con Y. Se X vuole parlare con Z, avrà bisogno di una chiave diversa. Ma è anche possibile decidere di avere una chiave comune (chiave di gruppo) che è comune a X, Y e Z, ma altre persone che fanno parte del gruppo, ma non sono la destinazione di un messaggio, sarebbero in grado di leggerlo.



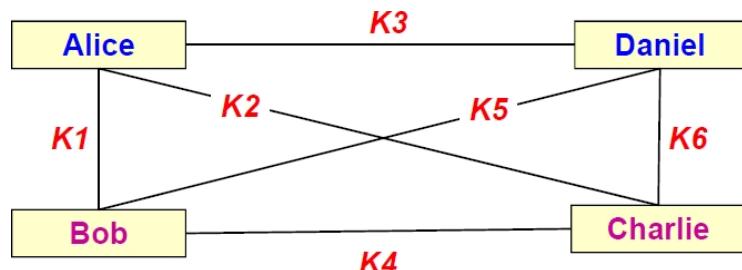
Distribuzione delle chiavi per la crittografia simmetrica

A completo a coppie

privata tra N parti richiede

$\frac{N \times (N-1)}{2}$ che è uguale alla chiave

diagonali di un poligono con N vertici. Se N diventa più grande, anche il numero di chiavi diventa molto più grande. Il problema è come fornire in modo sicuro le chiavi alle parti comunicanti. Le soluzioni sono:



- **OOB (Out-Of-Band)**: la chiave viene fornita alle parti che non utilizzano lo stesso canale elettronico usato per trasmettere il messaggio (la comunicazione diretta sarebbe la soluzione migliore per la condivisione della chiave, ad esempio sussurrando la chiave a qualcuno);
- **Mediante algoritmi di scambio di chiavi**: ne parleremo [più avanti](#).

Lunghezza delle chiavi segrete

La lunghezza della chiave segreta deve essere adeguata. Adeguata significa che se:

- l'algoritmo di crittografia era ben progettato e
- le chiavi di lunghezza N_{bit} sono tenute segrete;

allora l'unico attacco possibile è **quello a forza bruta (esaustivo)** che richiede $T = 2^{N_{bit}}$ prove (numero di combinazioni possibili). La tabella sul riassumere ciò che attualmente è considerato sicuro o insicuro. Qualsiasi chiave inferiore a 128 bit **non è sicura**. Ciò è dovuto

	symm.	40	64	128	256	...
	asymm.	512	1024	2048	4096	...
	<i>low security</i>					
	<i>high security</i>					

alla velocità della CPU attuale e al fatto che è possibile parallelizzare la ricerca della chiave (ad esempio, più computer che lavorano su una serie specifica di chiavi). La seconda riga si riferisce alle **chiavi**

asimmetriche. Ogni chiave asimmetrica inferiore a 2048 bit non è sicura. Non ci sono bordi nella colonna, ma una freccia che si espande sempre verso destra. L'area di scarsa sicurezza aumenta di anno in anno perché ogni anno i computer sono più potenti. Un altro problema è **per quanto tempo i dati rimarranno segreti**. Se i segreti devono rimanere tali, è importante

evitare la freccia di bassa sicurezza e rimanere sempre nella parte destra della tabella, ma allo stesso tempo più aumenta la dimensione della chiave, più gli algoritmi diventano lenti e non si tratta di un semplice raddoppio. Per gli algoritmi simmetrici il raddoppio della chiave dimezza le prestazioni, ma per gli algoritmi asimmetrici il raddoppio della chiave comporta un fattore da 4 a 10 di riduzione delle prestazioni.

Le sfide del DES

L'RC2 e l'RC4 erano algoritmi venduti privatamente, ma il creatore non ottenne molti soldi. Per questo motivo, il creatore iniziò una campagna per dimostrare che il DES era insicuro. Ha fissato un prezzo di 10.000 dollari per la 1st persona che fosse in grado di decifrare un messaggio cifrato DES utilizzando la forza bruta. Ciò significa $2^{56} = 72.057.594$ miliardi di chiavi possibili.

La sfida DES è iniziata il 18 febbraio 1997 ed è terminata il 17 giugno 1997. Dopo 4 mesi qualcuno ha scoperto il messaggio criptato utilizzando 15.000 computer in rete e dopo aver provato circa il 25% delle chiavi, 17.731.000, la chiave è stata trovata.

La sfida DES II è iniziata il 13 gennaio 1998 ed è terminata il 23 febbraio 1998. Qualcuno l'ha scoperto in un mese usando 20.000 computer e utilizzando l'87% (63.686.000) delle chiavi possibili.

Non è il numero di computer utilizzati a fare la differenza, ma la potenza del computer che in un anno è cambiata.

La sfida Des III è iniziata il 13-lug-98 ed è terminata il 15-lug-98. In due giorni è stato provato il 25% (17.903.000) delle chiavi. Ciò significa che in una sola settimana è stato possibile decifrare qualsiasi messaggio con crittografia DES. Questa volta è stato utilizzato solo un sistema speciale (DEEP CRACK), sviluppato dall'EFF al costo di 250.000 dollari. Hanno creato questo scopo speciale per dimostrare al governo (che utilizzava il DES) che non era sicuro.

Questa è la dimostrazione che è possibile costruire un sistema informatico in grado di decriptare un generico messaggio DES, ma:

- È necessario conoscere il tipo di dati (ad esempio, ASCII), altrimenti è necessario provare tutti i diversi tipi di formato per i file. (nel concorso si sapeva che si trattava di un messaggio di testo scritto in ASCII);
- La macchina non può decifrare i messaggi 3DES;
- Il DES non è intrinsecamente debole: utilizza solo una chiave breve.

La sfida DES IV è iniziata il 18 gennaio 1999 ed è terminata dopo 22 ore. Il 22% delle chiavi è stato provato utilizzando la macchina DEEP CRACK e alcune migliaia di workstation. La potenza di picco è stata di 250 Gkey/s mentre la potenza media è stata di 199 Gkey/s.

Dopo queste sfide, l'IETF modifica tutte le RFC consigliando di non usare il DES e suggerendo l'uso del 3DES. Esiste una specifica RFC-4772 che contiene le implicazioni di sicurezza dell'uso del DES. Una banca tedesca è stata condannata per una frode realizzata con un sistema che sfruttava la debolezza del DES. Il 15 gennaio 1999 il FIPS ha ritirato il DES (46/2) e lo ha sostituito con il 3DES (46/3). Contemporaneamente, il governo degli Stati Uniti ha avviato una gara per la selezione di un nuovo algoritmo simmetrico: **AES (Advanced Encryption Standard)** con lunghezza della chiave fino a 256 bit e dimensione del blocco di almeno 128 bit. C'erano 15 candidati ma solo 5 finalisti.

Il vincitore è stato annunciato il 2nd ottobre 2000, con il nome di RIJNDAEL e l'algoritmo è stato pubblicato nel novembre 2001 come FIPS-197. Due note per questo algoritmo:

- Non è il migliore per tutto, ma è il migliore per molti tipi di sistemi (ad esempio, i sistemi embedded).
- RIJNDAEL è stato l'unico algoritmo in cui l'America non è stata coinvolta (per evitare qualsiasi accusa di manipolazione).

Anche se l'algoritmo è stato pubblicato nel 2001, è stato gradualmente adottato dopo il 2010. Ci vuole così tanto tempo perché gli algoritmi di crittografia sono come il vino: i migliori sono quelli che invecchiano per

diversi anni.

Crittografia a chiave pubblica / asimmetrica

Oggi esiste un secondo tipo di crittografia inventato di recente (anni '70 del secolo scorso) e denominato **crittografia pubblica**: in questo tipo di crittografia, la **chiave-1 ≠ la chiave-2**. Se una viene usata per la crittografia, l'altra deve essere usata per la decriptografia, ma il ruolo può essere scambiato. Se una viene usata per la cifratura, l'altra deve essere usata per la decifrazione, ma il ruolo può essere scambiato. È anche chiamato **algoritmo asimmetrico** e le chiavi sono utilizzate a **coppie**. Poiché non esiste una sola chiave, le chiavi sono denominate in base al modo in cui vengono memorizzate: una viene mantenuta privata (**chiave privata**), mentre l'altra è pubblica (**chiave pubblica**). Questo tipo di

La crittografia ha un elevato carico computazionale, quindi di solito viene utilizzata per distribuire chiavi segrete e per creare firme elettroniche (con l'hashing) e non per memorizzare dati. Gli algoritmi principali sono: *Diffie-Hellman, RSA, DSA, El Gamal, ...*

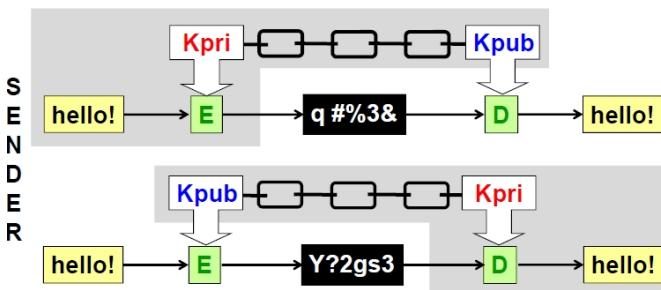


Figura 17 Come una coppia di chiavi può essere utilizzata nella crittografia asimmetrica di una comunicazione: le chiavi sono generate a coppia: **chiave privata** (Kpri) e **chiave pubblica** (Kpub). Le chiavi hanno una **funzionalità inversa**: i dati crittografati con una chiave possono essere decifrati solo dall'altra chiave.

Nella parte superiore, un messaggio viene crittografato con una chiave privata e inviato al destinatario, che utilizzerà la corrispondente chiave pubblica per decifrare il messaggio.

R
E
C
E
I
V
E
R

In genere, è il mittente che usa la sua chiave privata per criptare qualcosa. Tutte le altre persone al mondo conosceranno la chiave pubblica e saranno in grado di decifrare il messaggio.

In fondo, c'è il caso in cui qualcuno utilizzi la chiave pubblica della destinazione, perché solo la destinazione avrà la chiave privata corrispondente.

Queste due modalità di utilizzo di una coppia di chiavi sono utilizzate per ottenere due diverse funzionalità, descritte più dettagliatamente di seguito.

Firma digitale

Quella che segue è solo l'**idea** della firma digitale. La firma digitale segue il caso superiore dell'immagine precedente, la **crittografia dei dati effettuata con la chiave privata dell'autore**. X critta un documento con la sua chiave privata. Dal momento che è stato crittografato con la chiave privata, chiunque può utilizzare la chiave pubblica per decifrarlo. Quindi, non si tratta di un messaggio **segreto**, ma è "firmato" dal mittente: se qualcuno usa un altro messaggio pubblico, il mittente può essere "firmato".

non sarà possibile decifrare il messaggio. Questa è una prova ed è legalmente valida che alcuni bit sono stati creati da una persona. Se il messaggio è troppo grande, ci vorrà troppo tempo per cifrarlo: per questo motivo, i dati non vengono cifrati direttamente, ma solo il loro riassunto (*digest*). In questo modo si ottiene l'**autenticazione (e l'integrità) dei dati**.

Tuttavia, questa NON è la vera implementazione della firma digitale, che verrà spiegata completamente [in seguito](#).

Riservatezza senza segreti condivisi

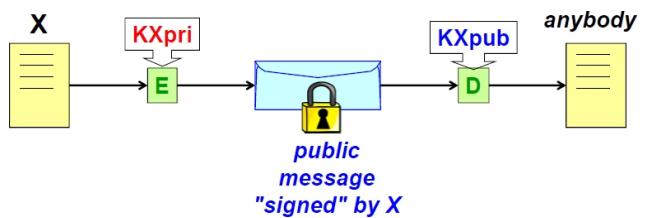


Figura 18 Utilizzo della crittografia asimmetrica per la firma di un messaggio

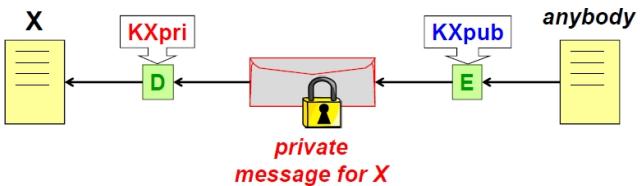


Figura 19 Utilizzo della crittografia asimmetrica per segnalare un messaggio

È possibile generare un **messaggio segreto** per un destinatario, data solo la sua chiave pubblica. Il messaggio viene crittografato con la chiave pubblica e solo il destinatario sarà in grado di decifrarlo utilizzando la chiave privata. Non è necessario condividere la chiave tra mittente e destinatario. Se si tratta di un messaggio di grandi dimensioni, la cifratura/decifratura richiederebbe molto tempo.

Algoritmi a chiave pubblica

I principali algoritmi utilizzati al giorno d'oggi sono due: **RSA** (*Rivest - Shamir - Adleman*) e **DSA** (*Digital Signature Algorithm*).

RSA consiste nel calcolare il **prodotto di due numeri primi**. Se si è l'attaccante, si vede il risultato e se si vuole attaccare il sistema si deve scoprire quali sono i due fattori. Sono quindi necessarie chiavi molto lunghe, perché se il risultato è 21 sarà facile da attaccare: 3 e 7 saranno i numeri primi. Il punto è che non esistono formule per calcolare tutti i numeri primi, per questo motivo più grande è il numero che si considera, più difficile è trovare i numeri primi, quindi più complesso è il problema per l'attaccante. RSA può essere utilizzato per implementare entrambe le funzionalità: segretezza e firma digitale. È brevettato solo negli Stati Uniti da RSA, ma il brevetto è scaduto il 20/09/2000 (non ci sono problemi di royalties).

Il DSA è il principale concorrente della firma digitale. Questo algoritmo sfrutta un problema matematico simile: **dati una base e un esponente, eseguire la potenza**. Se si è l'attaccante, si vede il risultato, ma non è facile sapere quali siano la base e l'esponente di quel numero. È come fare il logaritmo di un numero senza avere la base. Il problema del DSA è che può essere usato solo per creare una firma digitale, perché all'interno dell'algoritmo c'è una funzione di compressione a perdita unidirezionale: perdendo informazioni, il testo in chiaro originale non può essere recuperato. Questo è stato fatto di proposito perché il DSA è stato progettato dal governo degli Stati Uniti che non voleva dare ai cittadini la possibilità di nascondere qualcosa. Per ottenere la riservatezza senza segreti condivisi è necessario utilizzare l'**algoritmo El-Gamal** (con lo stesso principio del DSA). Il DSA è uno standard NIST per il DSS (FIPS-186).

Ottimizzazione computazionale RSA

Di solito, tutte le chiavi pubbliche hanno $E = 3, 17 \text{ or } 65537$ (0x10001, il numero di Fermat). La "E" sta per esponente e questi numeri sono stati scelti perché hanno solo due bit a "1" e tutti gli altri a "0", perché più bit vengono portati a 1, più operazioni devono essere fatte. In questo modo si ottiene un'elevata velocità di crittografia.

(quando si effettua la confidenzialità senza condividere i segreti) e l'alta velocità nella verifica della firma. Il rovescio della medaglia è che le altre operazioni sono lente (decrittazione, creazione della firma). Poiché questi valori sono comuni, alcune aziende hanno iniziato a sviluppare algoritmi ottimizzati per questi casi speciali, ma la conseguenza è che l'algoritmo è deoptimizzato per tutti gli altri valori dell'esponente.

Questo ha portato a due famosi attacchi contro Microsoft, perché usavano uno di questi algoritmi "speciali" e l'attacco creava una firma che aveva molti 1 nell'esponente e poi la dava a Microsoft Server (e questo creava una sorta di loop al server, perché l'elaborazione richiedeva molto tempo).

La lunghezza appropriata delle chiavi pubbliche è di 2048 bit, che offre un livello di sicurezza adeguato per diversi anni. Questo è dimostrato dalle sfide RSA con alcuni premi. Nel 2012 MS ha applicato una patch ai propri sistemi per non accettare più chiavi RSA < 1024. Lo stesso vale per Mozilla, che dal 2013 non accetta chiavi RSA < 2048 e MD5.

Sfide RSA

- **Sfide risolte (vecchio stile, dimensioni in cifre decimali)**
 - 10-apr-1996, RSA-130, 1000 anni MIPS
 - 22-agosto-1999, RSA-155 (512 bit), 8000 MIPS-anni
 - 9-maggio-2005, RSA-200 (663 bit), ~75 anni Opteron 2.2 GHz
 - 28-feb-2020, RSA-250 (829 bit), 2700 core-anni Xeon 2,1 GHz
- **Sfide risolte (nuovo stile, dimensioni in bit)**
 - 3-dic-2003, RSA-576 (174 cifre decimali)
 - 2-nov-2005, RSA-640 (193 cifre decimali)
 - 12-dic-2009, RSA-768 (232 cifre decimali) ~ 1500 anni Opteron 2,2 GHz; registrazione il 5/10/2019
- I premi sono stati ritirati dopo il 2007, l'interesse si è spostato su altri problemi di crittografia.

Scintillio

Ogni anno si tiene una conferenza chiamata [Eurocrypt](#), che è una conferenza sulla crittografia in Europa. Uno scienziato dei Laboratori RSA ha partecipato alla cosiddetta Ramp Session (significa che si mostrano idee interessanti e promettenti). Il professor Adi Shamir (da cui deriva la "S" di RSA) ha mostrato un

hardware insolito chiamato "**TWINKLE**" (acronimo di The Weizmann Institute Key Locating Engine), un computer elettro-ottico che implementa l'algoritmo di setacciatura per trovare i numeri primi. Questa macchina Twinkle

dovrebbe essere in grado di trovare i fattori di una chiave RSA con una velocità di due o tre ordini di grandezza superiore a quella di un PC veloce convenzionale. Il dispositivo funziona a una velocità di clock molto elevata (10 GHz), deve attivare i LED a intervalli di tempo precisi e utilizza una tecnologia su scala wafer. Secondo le stime del professore, il dispositivo può essere fabbricato con un costo di circa \$5.000. Shamir non è mai tornato alla conferenza successiva per mostrare il dispositivo.

Twirl (Il localizzatore di relazioni dell'Istituto Weizmann)

È un dispositivo elettronico per la fattorizzazione di numeri interi di grandi dimensioni. Implementa la fase di setacciamento dell'algoritmo di fattorizzazione di interi Number Field Sieve, che in pratica è la fase più costosa della fattorizzazione. TWIRL è più efficiente di diversi ordini di grandezza rispetto ai progetti precedenti, grazie all'elevata parallelizzazione algoritmica combinata con l'adattamento ai vincoli tecnologici dell'hardware. Sebbene sia abbastanza dettagliato, il progetto rimane ipotetico poiché il dispositivo non è stato costruito. TWIRL è costruito utilizzando l'attuale tecnologia VLSI e sarà possibile fattorizzare numeri interi a 1024 bit e quindi decifrare chiavi RSA a 1024 bit in un anno, al costo di qualche decina di milioni di dollari.

Firma del firmware per le calcolatrici TI

Questi fatti dimostrano che RSA non è totalmente sicuro: ci sono prove che prima o poi i fattori verranno trovati e il problema matematico risolto. Un esempio è la calcolatrice grafica TI83+, che ha un firmware protetto da una firma RSA a 512 bit. La chiave della firma è stata fattorizzata nel luglio 2009 dopo 73 giorni di calcolo su un PC Athlon64 dual-core a 1,9GHz. Nel 1999 lo stesso calcolo avrebbe richiesto 8000 MIPS-anni + Cray C916. Ora tutti gli utenti possono modificare autonomamente il firmware. Un firmware firmato significa che un software è firmato con una firma digitale (che è un pezzo di software). Se qualcuno altera il software, la firma digitale fallisce e c'è un controllo hardware che non funziona se il software viene modificato.

Distribuzione delle chiavi per la crittografia asimmetrica

La chiave privata non deve essere mai divulgata e deve essere protetta. L'altra chiave non ha bisogno di essere protetta, essendo una chiave pubblica deve essere distribuita il più possibile. Ma il problema è: **chi garantisce il legame (corrispondenza) tra la chiave pubblica e l'identità della persona?**

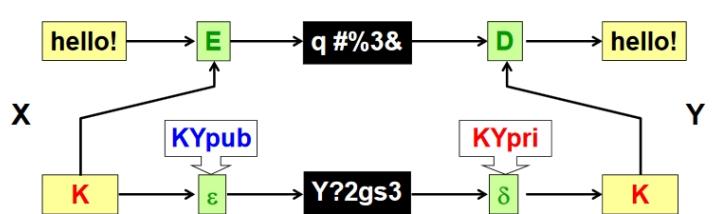
Esistono due soluzioni per questo problema:

- 1) **Scambio di chiavi OOB** (Out-Of-Band) (ad esempio, key party!)
- 2) La distribuzione della chiave pubblica avviene tramite una specifica struttura di dati denominata **certificato di chiave pubblica** (= certificato digitale). Ma qual è il formato del certificato? Vi fidate dell'emittente del certificato?

Scambio di chiavi segrete con algoritmi asimmetrici

In linea di principio, la crittografia asimmetrica è in grado di garantire la riservatezza senza segreto condiviso ma, come già detto, è lenta e questo la rende applicabile a piccole quantità di dati. Per questo motivo, una pratica comune è quella di utilizzarla per inviare la chiave segreta scelta per un algoritmo simmetrico, risolvendo allo stesso tempo il problema della distribuzione della chiave simmetrica degli algoritmi simmetrici e della lentezza di quelli asimmetrici per i big data.

Nella figura di esempio, X vuole inviare un messaggio cifrato a Y: per farlo, produce una chiave K (lunga di solito 128-256 bit) e la critta utilizzando la chiave pubblica di Y (KYpub), ottenendo ε: si noti che, data la piccola dimensione di K, l'esecuzione della sua crittografia asimmetrica è sufficientemente veloce. A questo punto Y utilizza la sua chiave privata



per decrittare ε , ottenendo la chiave simmetrica originale K , che sarà utilizzata per il resto della comunicazione tra X e Y , nel caso in cui la chiave simmetrica sia stata usata per la comunicazione tra X e Y .

ad esempio per inviare "hello". In genere, la chiave K può essere cambiata per ogni messaggio o per ogni sessione di rete; tuttavia, non è pensata per essere a lungo termine (ad esempio, anni). Questo tipo di crittografia viene applicato ogni volta che viene creata una connessione al server.

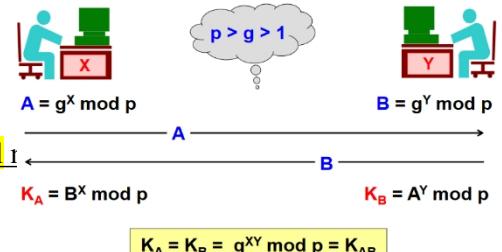
Figura 20 Utilizzo della crittografia asimmetrica per condividere una chiave simmetrica con riservatezza

Questo schema è sicuro? In alcuni ambienti, il fatto che X decida quale chiave sarà usata per la comunicazione effettiva non è accettabile: Y potrebbe sostenere che la chiave di X non è sicura, forse è stata divulgata e qualcuno potrebbe essere in grado di decifrare il messaggio nella comunicazione anche senza violare la crittografia asimmetrica utilizzata per lo scambio della chiave. Lo stesso ragionamento potrebbe applicarsi al caso opposto, in cui è Y a decidere la chiave da utilizzare. In questo caso, la chiave può essere prodotta congiuntamente utilizzando l'algoritmo **Diffie-Hellman**.

Algoritmo Diffie-Hellman

Le due parti (chiamate X e Y in questo esempio) coinvolte nella comunicazione si accordano su due numeri interi pubblici p (un numero primo grande) e g (chiamato "generatore"), dato il vincolo $p > g > 1$

i valori x e y sono segreti



1. La lunghezza di una chiave di Diffie-Hellman è definita come il

numero di entrambi i numeri p e g sono valori pubblici e tipicamente $g =$

2, 3 or 5. Entrambe le parti decidono autonomamente un numero (casuale), che chiamiamo x per il peer X e y per il peer Y: X calcola il numero $A = g^x \text{ mod } p$, e Y calcola

$B = g^y \text{ mod } p$, quindi si scambiano questi valori in chiaro.

Quando

riceve i valori dalla controparte, X calcola $K_A =$

$B^x \text{ mod } p$ e Y calcola $K_B = A^y \text{ mod } p$: è facile vedere che

$K_A = K_B = g^{xy} \text{ mod } p = K_{AB}$, quindi le parti hanno deciso insieme il valore della chiave simmetrica da utilizzare, perché dipende sia da x sia da y .

Figura 21 Accordo su una chiave simmetrica con l'algoritmo Diffie-Hellman

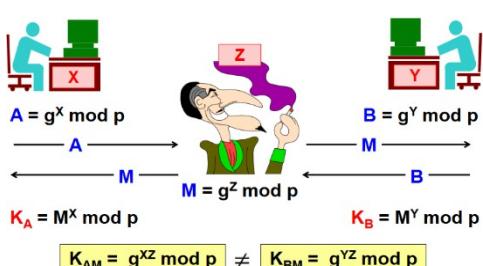
Dal punto di vista di un attaccante, anche conoscendo i valori di p e g , e quindi i valori A e B che vengono scambiati (ad esempio, cercando di eseguire un **man-in-the-middle passivo**), non è possibile calcolare K_{AB} , poiché dovrebbe conoscere anche x o y . Quindi l'algoritmo Diffie-Hellman è resistente agli attacchi di sniffing; tuttavia, è vulnerabile agli **attacchi di man-in-the-middle**.

attacchi **in-the-middle** se l'attaccante può manipolare i dati: questa manipolazione può essere rilevata se i valori scambiati sono autenticati. In questo caso, sono necessari certificati per le chiavi Diffie-Hellman, oppure esiste una variante dell'algoritmo chiamata **Diffie-Hellman autenticato** (o MQV, dal nome degli scienziati che lo hanno inventato, Menezes-Qu-Vanstone).

Poiché utilizza solo informazioni pubbliche, viene definito il primo algoritmo a chiave pubblica inventato, ed è spesso utilizzato per concordare una chiave segreta, per cui viene definito **algoritmo di accordo di chiave**, in contrapposizione all'**algoritmo di distribuzione di chiave** appena visto. È stato brevettato negli Stati Uniti, ma è libero da royalties dal 29 aprile 1997.

DH: attacco man-in-the-middle

Come detto, l'unico modo per attaccare la Diffie-Hellman dopo aver eseguito lo sniffing è eseguire un **attacco di forza bruta** per trovare $x = \log_g A \text{ mod } p$, che è impossibile da risolvere poiché è un problema logaritmico discreto. Nel caso di un attaccante che esegue un man-in-the-middle attivo, il risultato è diverso.



Supponiamo che un man in the middle Z partecipi alla comunicazione e che, come X e Y, calcoli $M = g^z \text{ mod } p$: quando il peer X invia il suo valore A , Z lo ferma e invia in cambio il valore M . Analogamente, quando il peer Y invia il suo valore B , lo interrompe e invia lo stesso valore M . Così facendo, il peer X calcolerà $K_A = M^x \text{ mod } p$, mentre Y calcolerà $K_B = M^y \text{ mod } p$: questa volta i due valori non sono uguali, infatti $K_{AM} = g^{xz} \text{ mod } p$ mentre $K_{BM} = g^{yz} \text{ mod } p$. In realtà stanno calcolando una chiave condivisa

Figura 22 Un esempio di attacco attivo man-in-the-middle a una comunicazione che utilizza

con l'uomo nel mezzo, e cifreranno i dati nel mezzo, il quale ispezionerà i dati, li modificherà e li cifrerà che possono essere decifrati dall'uomo di nuovo per la destinazione, con l'altra chiave condivisa.

Questo attacco è reso possibile dal fatto che quando un peer riceve un valore, non ha alcuna prova di chi lo ha inviato: questo è il motivo per cui una possibile soluzione è l'uso di [certificati](#). Un'applicazione di DH potrebbe essere quella dell'HTTPS: il browser potrebbe produrre una chiave e inviarla crittografata con la chiave pubblica del server, oppure il server potrebbe pubblicare

il suo valore DH firmato con la sua chiave privata e lo distribuisce. Il DH è molto usato oggi per concordare una chiave e c'è anche la possibilità di avere un'autenticazione (perché altrimenti è possibile un attacco MITM).

Brute-forcing Diffie-Hellman: crittografia contro i computer quantistici

Il DH ha la complessità del logaritmo discreto, il modo migliore per trovare la soluzione $x = \log_g A \bmod p$, è eseguire prodotti successivi, quindi la complessità è lineare in p , quindi esponenziale nel numero di bit di p . Esistono anche altri algoritmi, ma non sono comunque polinomiali in termini di tempo.

Tuttavia, è possibile utilizzare l'algoritmo di Shor, la cui complessità è $O(\log(N)^3)$, che è in grado di risolvere RSA e DH, ma richiede un computer quantistico. Il calcolo quantistico è al momento una tecnologia sperimentale: nel 2012 un computer quantistico con 7 qubit ha calcolato la fattorizzazione di $21 = 7 \times 3$, e nel 2014 un computer quantistico adiabatico ha fattorizzato $56153 = 233 \times 241$. Questi risultati sono notevoli, ma ancora lontani

abbastanza dai grandi numeri utilizzati al giorno d'oggi. Poiché le firme e i segreti devono resistere per molti anni (circa 10-30 anni), ci sono ragioni per essere preoccupati che i computer quantistici diventino abbastanza potenti da rompere questi meccanismi di sicurezza: gli esperti stanno iniziando a cercare altri problemi matematici che non possono essere attaccati da un computer quantistico.

Un nuovo tipo di crittografia di questo tipo è l'**ECC** (*Elliptic Curve Cryptosystem*). Il concetto è il seguente: invece di utilizzare l'aritmetica modulare, le operazioni vengono eseguite sulla superficie di una curva 2D (ellittica). Non tutti i punti dello spazio cartesiano sono punti validi, ma solo quelli che soddisfano l'equazione della curva 2D. Il problema del logaritmo discreto su una curva di questo tipo è che, poiché la rappresentazione dei punti è più complessa, in generale il problema è più complesso di quello della normale aritmetica modulare. Ciò offre l'opportunità di utilizzare chiavi più corte di 1/10. La maggior parte degli algoritmi sono stati rivisitati per adattarli alla curva ellittica (RSA non può essere adattato):

- **ECDSA** per la firma digitale:
 - messaggio digerito calcolato con una normale funzione di hash (ad esempio, SHA-256);
 - firma = coppia di scalari derivati dal digest più alcune operazioni sulla curva (si veda più avanti a proposito delle operazioni sulla curva).
- **ECDH** per l'accordo sulla chiave;
- **ECMQV** per l'accordo di chiave autenticato;
- **ECIES** (EC Integrated Encryption Scheme) per la distribuzione delle chiavi:
 - genera una chiave di crittografia simmetrica (ad esempio una AES-128) con operazioni sulla curva;
 - fornisce al destinatario le informazioni (basate sulla sua chiave pubblica) necessarie per ricompilare la chiave di crittografia.

Come funziona

Il primo elemento da selezionare in un algoritmo di EC è la curva stessa, che deve essere della forma di $y^2 = x^3 + ax + b$ con $4a^3 + 27b^2 \neq 0$, quindi si selezionano due punti P, Q appartenenti alla curva, quindi il punto $R = (x, y) = P + Q$, con $x = \lambda^2 - x_p - x_Q$ e $y = \lambda(xP - x) - yP$, dove:

- $\lambda = (y_p - y_Q)(x_p - x)_Q$ *if $P \neq Q$ quindi può essere utilizzato per calcolare P+Q;*
- $\lambda = (3x_p + a)y_p$, *if $P = Q$ quindi può essere utilizzato per calcolare 2P.*

Date queste formule, solo l'addizione di due punti e la moltiplicazione di un punto per uno scalare sono definite e utilizzate dagli algoritmi EC.

EC-Diffie-Hellman

Con riferimento al DH originale, ora i peer non concordano su p e g , ma concordano sulla stessa curva ellittica (stesso insieme di parametri a e b) e su un punto G appartenente alla curva. Poi A sceglie un valore casuale x e calcola $X = xG$; analogamente, B calcola $Y = yG$. Poi A e B si scambiano questi valori e ciascuno esegue la moltiplicazione del valore ricevuto per il valore scelto a caso in precedenza, per cui A calcola $K = xY$ e B calcola $K' = yX$. Come nella DH semplice, vale che $K = K' = xyG$. L'operazione è quindi più semplice

rispetto alla DH semplice, ma la complessità per l'attaccante si basa sul calcolo dei valori x o y conoscendo X o Y .

Caso storico: Sony PS3 hacking (Console Hacking 2010, PS3 Epic Fail)

La PS3 di Sony è una macchina Linux su cui l'accesso diretto al sistema è bloccato da Sony, così come qualsiasi modifica ad esso mediante la firma digitale del firmware da parte di Sony stessa. La PS3 è dotata di Linux incorporato con loader che verifica la firma ECDSA dei binari prima dell'esecuzione. La generazione di una firma ECDSA richiede un nonce casuale, altrimenti dalla firma si può calcolare la chiave privata. Il problema è che Sony ha deciso un numero casuale *fisso* (quindi tecnicamente non è più un nonce) e di conseguenza la chiave privata è stata calcolata e distribuita in tutto il mondo, in modo che chiunque possa eseguire i propri binari sulla propria PS3.

Sony ha cercato di portare gli hacker in tribunale, accusandoli di aver violato il loro software: il giudice ha respinto l'accusa perché la misura di sicurezza non era stata implementata correttamente.

Integrità del messaggio

Finora abbiamo visto che esistono metodi per garantire che un messaggio da un mittente a un destinatario non possa essere compreso da terzi. Tuttavia, è ancora possibile per un aggressore intercettare il traffico tra le due parti e modificare il messaggio criptato: anche se questo non gli dà la possibilità di alterare il messaggio in modo specifico, può modificarlo in modo imprevedibile (ad esempio capovolgendo alcuni bit del messaggio). In alcuni tipi di comunicazione questo fa sì che il messaggio non sia più comprensibile quando viene decriptato dal destinatario, in modo da rendere evidente che qualcosa è andato storto durante la trasmissione: in questo caso il destinatario può chiedere la ritrasmissione, e un attacco continuo come questo è come minimo un DoS. In altri casi, ad esempio quando il messaggio è un numero di conto corrente bancario, è molto probabile che il risultato sia un numero diverso, ma comunque valido, per cui la destinazione non ha modo di verificare se è sbagliato. Questo è particolarmente pericoloso quando la comunicazione avviene tra sistemi automatici.

Insieme alla riservatezza, l'integrità è quindi una questione molto importante. L'integrità non significa impedire la modifica dei dati, ma significa che quando si ricevono dei dati, il ricevente può verificare se questi sono stati modificati o meno. Questo è importante non solo nelle comunicazioni, ma anche per la memorizzazione dei dati sul disco: può essere importante verificare se qualcuno ha modificato i dati sul disco da quando sono stati scritti. L'importanza dell'integrità è dimostrata da uno studio condotto dall'esercito britannico, che ha stimato che solo il 10% delle comunicazioni necessitava di riservatezza, ma il 100% di esse di integrità. L'integrità implica anche l'autenticazione, poiché un messaggio che non è stato modificato è anche autentico (il che significa che è lo stesso inviato dal mittente).

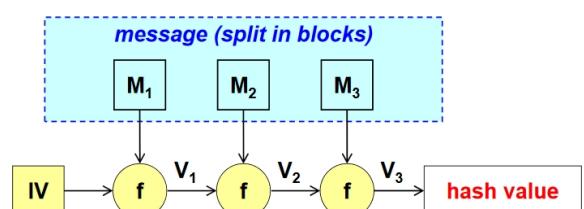
Per garantire l'integrità, la tecnica utilizzata è quella di calcolare un **digest**, cioè un *riassunto a lunghezza fissa* dell'intero messaggio da proteggere. Un digest è concettualmente simile a un checksum (ampiamente utilizzato nelle comunicazioni per rilevare gli errori di trasmissione); tuttavia, gli algoritmi utilizzati per calcolare un checksum sono facilmente attaccabili, per cui un digest viene solitamente calcolato tramite una **funzione di hash crittografico**. Il digest deve essere:

- Veloce da calcolare;
- Impossibile o molto difficile da invertire, cioè tornare dal digest ai dati originali;
- Dovrebbe essere difficile creare "collisioni" (due messaggi diversi non dovrebbero produrre lo stesso digest).

Funzioni hash crittografiche

La funzione hash funziona in questo modo:

- divide il messaggio M in N blocchi $M_1 \dots M_N$;
- applicare iterativamente una funzione base (f);
- $v_k = f(v_{k-1}, M_k)$ con $v_0 = IV$ e $h = VN$. IV è



un valore di inizializzazione che non necessita di
essere annunciato o essere casuale. In genere, è fisso e
specificato all'interno dell'algoritmo stesso (la sorgente
e la destinazione devono eseguire lo stesso calcolo)

Figura 23 Come funziona una funzione hash crittografica

Famiglia SHA-2

name	block	digest	definition	notes
MD2	8 bit	128 bit	RFC-1319	obsolete
MD4	512 bit	128 bit	RFC-1320	obsolete
MD5	512 bit	128 bit	RFC-1321	obsolete
RIPEMD-160	512 bit	160 bit	ISO/IEC 10118-3	good
SHA-1	512 bit	160 bit	FIPS 180-1	sufficient
SHA-224	512 bit	224 bit		
SHA-256	512 bit	256 bit	FIPS 180-2	good
SHA-384	512 bit	384 bit	FIPS 180-3	
SHA-512	512 bit	512 bit		
SHA-2				
SHA-3	1152-576	224-512	FIPS 202 FIPS 180-4	excellent

Gli algoritmi più utilizzati al giorno d'oggi sono quelli della **famiglia SHA-2**: hanno la stessa dimensione del blocco ma una dimensione del digest sempre maggiore. Poiché ereditano la struttura di SHA-1, che è stata violata e non dovrebbe più essere utilizzata, è possibile che presto non siano più sufficientemente sicuri, per cui SHA-3 è già stato sviluppato e dovrebbe essere la scelta da adottare una volta dimostrata la sua forza. La famiglia SHA-2 è stata una soluzione rapida dopo l'attacco SHA-1 ed è stata sviluppata allungando il digest. Gli algoritmi principali sono: SHA-256 che

utilizza una parola a 32 bit e SHA-512 utilizza una parola a 64 bit (rispettivamente per PC a 32/64 bit).

La lunghezza del digest è importante per evitare l'**aliasing**, cioè la collisione di due messaggi diversi sullo stesso digest: rompere un algoritmo di digest significa trovare un secondo messaggio che generi lo stesso digest del primo.

Se l'algoritmo è ben progettato e genera un digest di N bit, allora la probabilità di aliasing è $P_A \propto \frac{1}{2^{N \text{ bits}}}$

Questo è il motivo per cui sono necessari digest con molti bit (perché sono coinvolti eventi statistici).

Come conseguenza di considerazioni statistiche (si veda il "[paradosso del compleanno](#)"), un algoritmo di digest a N bit è insicuro.

quando vengono generati più di 2^N digest, perché la probabilità di avere due messaggi con lo stesso digest è $P_A \approx 50\%$. Se un attaccante sniffera la rete, può semplicemente memorizzare tutti i messaggi e i relativi digest che passano. Con molti messaggi e digest memorizzati la probabilità di avere gli stessi digest per due messaggi diversi è

alto. A questo punto è possibile scambiare i due messaggi, perché hanno lo stesso digest. Un crittosistema è "bilanciato" quando gli algoritmi di cifratura e di digest hanno la stessa resistenza: così, SHA-256 e SHA-512 sono stati progettati per essere utilizzati rispettivamente con AES-128 e AES-256, mentre SHA-1 (cioè SHA-160) si abbinava a Skipjack-80 (un algoritmo sviluppato dai servizi segreti britannici). Si noti che AES ha metà bit di

SHA (grazie a $a^{\frac{N}{2}}$).

Famiglia SHA-3

A causa dei problemi già citati con la famiglia SHA-2, SHA-3 è stato un concorso per una nuova funzione hash dedicata: il vincitore è stato annunciato il 2 ottobre 2012 ed è stato Keccak (autori = G.Bertoni, J.Daemen, G. Van Assche (STM), M.Peeters (NXP)). Il team del NIST ha elogiato l'algoritmo Keccak per le sue numerose e ammirabili qualità, tra cui il **design elegante e la capacità di funzionare bene su molti dispositivi informatici diversi**. La **chiarezza della struttura di Keccak** si presta a una **facile analisi** (durante il concorso tutti gli algoritmi presentati sono stati resi disponibili per essere esaminati e criticati dal pubblico), e Keccak ha **prestazioni superiori nelle implementazioni hardware** rispetto a SHA-2 o a qualsiasi altro finalista.

Keccak è utilizzato per definire diverse funzioni hash (FIPS- 202): esistono tre versioni di SHA-3, che sono destinate a sostituire le corrispondenti funzioni hash della famiglia SHA-2, perché producono la stessa forza. Inoltre, esistono lunghezze di uscita e hanno

function	output	block capacity	definition	strength
SHA3-224(M)	224	1152	Keccak[448](M 01, 224)	112
SHA3-256(M)	256	1088	Keccak[512](M 01, 256)	128
SHA3-384(M)	384	832	Keccak[768](M 01, 384)	192
SHA3-512(M)	512	576	Keccak[1024](M 01, 512)	256
SHAKE128(M,d)	d	1344	Keccak[256](M 1111, d)	min(d/2,128)
SHAKE256(M,d)	d	1088	Keccak[512](M 1111, d)	min(d/2,256)

Figura 24 Forza delle diverse

funzioni di hash appartenenti alla famiglia SHA-3 in funzione della lunghezza del digest e della dimensione del blocco

L'output a lunghezza variabile, poiché è possibile parametrizzare la lunghezza dell'output: ovviamente, un digest più breve comporta una protezione più debole.

Sono state definite altre varianti (NIST SP.800-185):

- cSHAKE (parametri di dominio personalizzabili);
- KMAC (keyed digest);
- TupleHash (hash di una tupla, dove la sequenza è importante);
- ParallelHash (hash veloce sfruttando il parallelismo interno della CPU).

KDF (funzione di derivazione delle chiavi)

Le funzioni hash crittografiche non servono solo per i digest, ma possono essere utilizzate anche per ricavare (creare) una chiave: una chiave crittografica deve essere casuale, cioè i bit 1 e 0 devono essere distribuiti uniformemente. Tuttavia, di solito gli utenti inseriscono password o frasi di accesso che sono indovinabili e non casuali. Questo accade perché è poco pratico per un utente ricordare una sequenza di bit senza senso. Di solito, una chiave è derivata da una password

o passphrase inserita dall'utente, in modo che $K = KDF(P, S, I)$, dove:

- P è la password o passphrase dell'utente;
- S è un sale, una variante imprevedibile che rende K difficile da indovinare anche conoscendo P ;
- I è il numero di iterazioni della funzione base (per rallentare il calcolo e rendere la vita complessa agli attaccanti).

Quando si forniscono informazioni alla destinazione, la passphrase viene trasmessa in modo sicuro, mentre il sale viene inserito nel messaggio stesso: il sale è qualcosa di simile a un IV, un numero casuale di grandi dimensioni utilizzato per evitare precomputazioni.

Esistono due principali funzioni di derivazione delle chiavi che si basano su funzioni hash crittografiche:

- **PBKDF2 (RFC-2898)** utilizza SHA-1, con $|S| \geq 64$ e $I \geq 1000$:

La funzione è $DK = PBKDF2(PRF, PWD, Salt, C, dkLen)$, dove:

- PRF = funzione pseudorandom di due parametri con lunghezza di uscita $hLen$ (ad esempio, un HMAC con chiave);
- PWD = password da cui viene generata una chiave derivata;
- $Salt$ = sale crittografico;
- C = numero di iterazioni desiderate;
- $dkLen$ = lunghezza desiderata della chiave derivata;
- DK = chiave derivata generata = $T_1 || T_2 || \dots ||_{hLen}^{ThLen}$ (where each $|T_i| = hLen$).

Un'importante applicazione di questa KDF è nelle reti wireless, in particolare nelle soluzioni *WPA2*.

Quando si inserisce la passphrase, si fa un uso implicito di questa funzione: $DK = PBKDF2(HMAC$

$SHA1, PWD, ssid, 4096, 256)$, dove $ssid$ è il nome della connessione wireless.

Tuttavia, PBKDF2 può essere attaccato perché C può essere grande, ma questo richiede solo molti calcoli ma non molta RAM, quindi può essere attaccato da ASIC o GPU. Una contromisura consiste nell'aumentare la RAM necessaria per l'attacco. Nel 2013 è stato lanciato un concorso pubblico a questo proposito e il 20 luglio 2015 il vincitore è stato "Argon2".

- **HKDF (RFC-5869)** utilizza HMAC.

Abbiamo visto che un digest può essere usato in coppia con un messaggio per ottenere l'integrità; in genere, la parte aggiunta al messaggio per l'integrità è chiamata **MIC** (Message Integrity Code), a causa del codice aggiunto al messaggio per proteggerne l'integrità. Se un messaggio non viene modificato, significa che è autentico e in questo caso il codice è chiamato **MAC** (Message Authentication Code). I due significati sono identici, ma il MIC è solitamente utilizzato nell'ambiente delle telecomunicazioni e delle reti, mentre il MAC è usato soprattutto nel campo delle applicazioni. Di solito, quando si parla di MIC o MAC, si presuppone anche che il digest sia codificato, come vedremo in seguito. Poiché questo codice è un'aggiunta al messaggio

originale, di solito viene incluso anche un identificatore univoco del messaggio per evitare *attacchi di replay*, chiamato **MID** (Message IDentifier). In questo contesto, un attacco replay non è un attacco all'integrità di un singolo messaggio, ma all'integrità del processo di trasmissione.



Protezione del digest

Il problema da affrontare ora è come proteggere il digest: nel caso semplice in cui un digest venga utilizzato per verificare che un file su disco non sia stato modificato, un buon approccio potrebbe essere quello di memorizzare il digest in un luogo diverso e sicuro, in modo che non venga alterato. Quando si inviano dati, il digest non può essere scambiato fuori banda, quindi significa che il digest deve essere intrinsecamente sicuro. Le tecniche maggiormente adottate sono due:

- Autenticazione mediante **digest a chiave**;
- Crittografia autenticata.

Autenticazione tramite digest a chiave

Il digest viene inviato insieme al messaggio, ma viene calcolato non solo sui dati ma anche su una chiave segreta, condivisa tra mittente e destinatario. Quindi, vengono eseguite queste operazioni:

- **Mittente:** $d = \text{digest}(K, M)$;
- **Trasmissione:** $M \parallel d$;
- **Ricevitore:** $d' = \text{digest}(K, M)$;
- **Verifica:** se $(d == d')$ allora OK altrimenti significa che qualcosa è stato modificato durante la trasmissione (i dati o il digest, non importa).

Questo tipo di schema:
non fornisce confidenzialità
non fornisce non ripudio
fornisce integrità
fornisce autenticazione

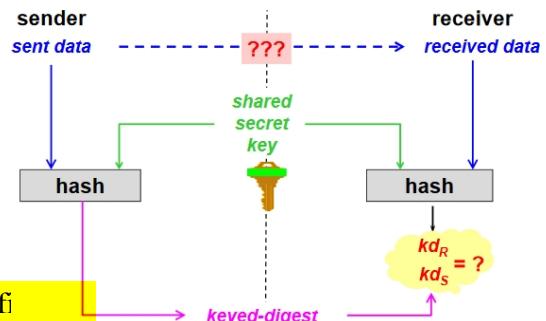


Figura 25 Schema del funzionamento di un digest con chiave

I vantaggi di questa tecnica sono che vi è una sola operazione (calcolo del digest) e che vengono aggiunti solo pochi dati. Vale la pena notare che questo meccanismo garantisce l'autenticazione, poiché solo chi conosce la chiave può confrontare il digest trasmesso con quello calcolato sui dati ricevuti. Questa è la soluzione più veloce per fornire integrità e autenticazione ai dati. Tuttavia, non fornisce il **non ripudio**: supponiamo che il destinatario di un ordine commerciale voglia portare in tribunale il mittente di quell'ordine perché ripudia l'ordine stesso, allora il destinatario dovrebbe mostrare come prova il messaggio insieme al digest che dimostra l'integrità e l'autenticità. In questo caso non è possibile affermare che il mittente ha originato l'ordine, poiché il destinatario ha tutti i mezzi per crearlo da solo: ha i dati e la chiave su cui calcolare un digest valido.

Questa soluzione garantisce quindi l'autenticazione, in quanto il destinatario può essere sicuro che i dati provengono dal mittente e non sono stati modificati, ma non fornisce di per sé il non ripudio, in quanto è impossibile per una delle due parti dimostrare a terzi che l'altra ha creato il messaggio.

Digest con chiave: possibili errori

Se $kd = H(K | M)$ (il che significa che si pre-applica la chiave al messaggio) allora è possibile modificare il messaggio aggiungendo alla sua fine uno o più blocchi, in modo tale che $kd' = H(K | M | M') = f(kd, M')$: questo significa che

La modifica dei dati con l'aggiunta di un ulteriore blocco è sufficiente per calcolare un nuovo cifrario senza conoscere la chiave.

Se $kd = H(M | K)$ allora è possibile modificare il messaggio aggiungendo prima di esso un blocco opportuno, tale che $kd =$

$$H(M' | M | K) \text{ choosing } M \text{ s.t. } IV = f(IV, M').$$

Le contromisure di protezione sono:

- inserisce nei dati digeriti anche la lunghezza di M ;
- definire $kd = H(K | M | K)$;
- utilizzare un **keyed digest standard**: il migliore keyed digest oggi ampiamente utilizzato è HMAC.

Digest con chiave:

HMAC è definito in RFC-2104 e utilizza una funzione di hash di base H. Quindi, HMAC non è una funzione di hash, ma è un modo per calcolare un MAC sicuro a partire da una funzione di hash H. La funzione di hash di base H deve avere una dimensione di blocco B, con un'uscita lunga L byte, con $B > L$. Seguono alcune definizioni:

- **ipad** = 0x36 ripetuto B volte; è un padding composto da 0x36 ripetuto più volte per riempire tutto il blocco.
- **opad** = 0x5C ripetuto B volte;
- **chiavi deprecate** s.t. $|K| < L$; non utilizzare mai una chiave più piccola dell'output. Ad esempio, se si utilizza HMAC con SHA-1, è necessaria una chiave di almeno 160 bit.
- se $|K| > B$ allora $K' = H(K)$ altrimenti $K' = K$; se la chiave è più grande del blocco, allora la dimensione della chiave viene ridotta calcolando l'hash della chiave (altrimenti viene utilizzato quello originale).
- se $|K'| < B$ allora K' è 0-padded fino a B byte;

dato che, **hmac = $H(K' \oplus opad | H(K' \oplus ipad | data))$** .

$L'K'$ viene sottoposto a XOR con l'ipad e viene preaggiunto ai dati. Quindi viene calcolato l'hash. Dopo l'hashing, $L'K'$ viene sottoposto a XOR con opad e viene preapplicato al risultato dell'hash e l'hash viene nuovamente calcolato. Esso

è facile capire che fondamentalmente *HMAC* è un doppio hash dei dati composto in qualche modo con la chiave. Questo è molto più forte della semplice pre- o post-applicazione della chiave ai dati. Questa è la soluzione più utilizzata quando è necessario creare un MAC (o MIC, è la stessa cosa). **L'HMAC è tipicamente utilizzato per proteggere l'integrità. Se è necessario proteggere anche la riservatezza, c'è CBC-MAC.**

CBC-MAC

A volte non abbiamo bisogno solo di integrità, ma anche di riservatezza, quindi di crittografia informatica. In questo caso, piuttosto che utilizzare un'altra funzione (ad esempio, stiamo usando AES per la crittografia e abbiamo bisogno anche di SHA-256 per l'integrità, cioè abbiamo bisogno di più codice, il che non è sempre possibile, ad esempio nei sistemi embedded), è possibile calcolare un MAC non basato su una funzione di hash ma su un algoritmo simmetrico: questo è chiamato **CBC-MAC**.

Sfrutta un algoritmo di crittografia simmetrica a blocchi, in modalità CBC con un IV nullo, prendendo come MAC l'ultimo blocco crittografato. Il messaggio M viene suddiviso in N blocchi $M_1 \dots MN$, quindi prendendo $v_0 = 0$ per $(k \neq \dots N)$ fare $v_k = enc(K, M_{k \oplus v_{k-1}})$; infine, $CBC-MAC = VN$. Quindi, la dimensione della CBC-MAC è pari alla dimensione di un blocco dell'algoritmo di crittografia simmetrica sottostante. Se ad esempio si utilizza AES-CBC-

MAC, allora il digest sarà lungo 128 bit. La versione migliore del CBC-MAC basato su DES è il *Data Authentication Algorithm* (standard FIPS 113, ANSI X9.17).

CBC-MAC è sicuro solo per i messaggi a lunghezza fissa, perché altrimenti gli attacchi che estendono la lunghezza del messaggio possono avere successo; per i messaggi a lunghezza variabile è meglio usare CMAC.

Integrità e segretezza: come combinarle?

Abbiamo detto che nella maggior parte delle applicazioni abbiamo bisogno sia di integrità che di riservatezza. Ciò può essere ottenuto attraverso due operazioni ortogonali: la crittografia simmetrica con una chiave K_1 per la **segretezza** e il key digest (MAC) con K_2 per l'**integrità**. Esistono vari modi per combinare queste operazioni:

1. **Authenticate-and-encrypt (A&E)**: prima si critta il testo in chiaro con K_1 , poi si concatena il risultato con il MAC calcolato sul testo in chiaro con K_2 : in formule, $enc(K_1, p) | mac(K_2, p)$. In questo modo nessuno può leggere il testo in chiaro perché è crittografato e nessuno può alterare il testo cifrato, perché dopo la decrittazione è possibile calcolare nuovamente il MAC e il

si noterà qualsiasi differenza.

Lo svantaggio principale è che per verificare l'integrità il messaggio deve essere prima decriptato (un'operazione che richiede tempo), perché il MAC è calcolato sul testo in chiaro, quindi c'è una

Attacco DoS perché se l'attaccante dovesse modificare anche un bit del messaggio e visto che è richiesto di decriptare prima il messaggio per verificare il MAC allora perderemo tanto tempo nel decriptare il messaggio (operazione lunga) per vedere poi che il messaggio è stato modificato e quindi se applicato a tutti i messaggi abbiano un DoS

vulnerabilità contro gli attacchi DoS. Inoltre, per lo stesso motivo, il MAC potrebbe far trapelare alcune informazioni sul testo in chiaro. Non è la soluzione migliore, ma è utilizzata dal protocollo SSH.

2. **Authenticate-then-encrypt (AtE):** prima viene calcolato il MAC con K_2 sul testo in chiaro, poi viene concatenato con il testo in chiaro e il risultato crittografato con $\{K_1\}$ in formule, $enc(K_1, p | mac(K_2, p))$. Per ragioni simili al caso precedente, questa soluzione è ancora vulnerabile al DoS attacchi in quanto è necessario decriptare l'intero messaggio prima di poterne verificare l'integrità mediante

calcolo del MAC. Tuttavia, questa soluzione è migliore della precedente perché non comporta la perdita di informazioni, in quanto il MAC non viene inviato in chiaro, ma crittografato.

Questa soluzione è utilizzata da SSL e TLS fino alla versione 1.2: TLS versione 1.3 utilizza un approccio completamente diverso.

3. **Encrypt-then-authenticate (EtA):** prima si critta il testo in chiaro con K_1 , poi si concatena il risultato con il MAC calcolato sul testo cifrato ottenuto nel passo precedente: in formule, $enc(K_1, p) | mac(K_2, enc(K_1, p))$. In questo modo è possibile evitare gli attacchi DoS, infatti un'alterazione del testo cifrato verrà notata immediatamente, cioè senza dover prima decifrare il messaggio. Questa è la soluzione utilizzata da IPsec.

Sicurezza di queste composizioni

È importante notare che una combinazione impropria di algoritmi sicuri può portare a un risultato insicuro!

- **Authenticate-and-encrypt (A&E):** insicuro se non eseguito in un unico passaggio;
- **Authenticate-then-encrypt (AtE):** sicuro solo con la crittografia CBC o a flusso;
- **Encrypt-then-authenticate (EtA):** è la modalità più sicura, ma bisogna fare attenzione agli errori di implementazione: ad esempio, bisogna sempre includere l'IV e il nome degli algoritmi nel calcolo del MAC, altrimenti sono possibili alcuni attacchi.

Poiché nessuna di queste tre soluzioni è perfetta, gli sforzi attuali sono rivolti a un algoritmo AE congiunto, denominato **crittografia autenticata**.

Crittografia autenticata

L'idea è quella di utilizzare un'unica operazione per la privacy e l'autenticazione (e l'integrità). Ciò comporterà i seguenti vantaggi:

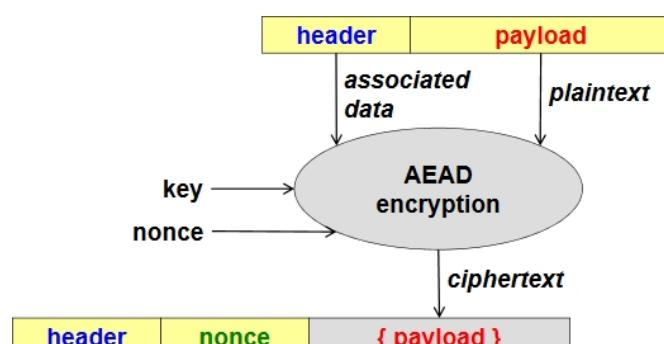
- una sola chiave e un solo algoritmo;
- velocità superiore;
- minore probabilità di errore nella combinazione delle due funzioni. Quest'ultimo aspetto è talvolta richiesto anche dalle applicazioni, che potrebbero avere requisiti diversi in termini di integrità e privacy. Analizziamo due esempi:
 - *pacchetti di rete:* non è possibile criptare l'intero pacchetto, infatti l'intestazione IP è necessaria per l'inoltro attraverso la rete, quindi il carico utile necessita di privacy e deve essere crittato, e l'intero pacchetto necessita di integrità;
 - *posta elettronica:* anche in questo caso valgono ragionamenti analoghi. Infatti, il corpo può essere crittato, ma ad esempio il campo "A:" non può esserlo, altrimenti il server di posta non saprà qual è la destinazione e quindi a chi deve inviare il messaggio.

Inoltre, ci sono altre motivazioni per un algoritmo di questo tipo, perché le normali modalità di crittografia sono soggette ad attacchi di tipo chosen-ciphertext quando vengono utilizzate on-line:

- l'attaccante modifica un testo cifrato;
- quindi osserva se il ricevitore segnala o meno un errore (ad esempio, un attacco **padding oracle** o **decryption oracle**).

Esistono vari modi per eseguire la crittografia autenticata, uno dei più importanti è la **crittografia autenticata con dati associati (AEAD)**, documentata nella RFC 5116.

Questo è esattamente il tipo di soluzione necessaria per la posta elettronica e i pacchetti di rete: in linea di principio la crittografia autenticata è un algoritmo che fornisce entrambe le funzionalità sopra citate su un



testo in chiaro. AEAD, al contrario, fa un

distinzione tra la parte che necessita di riservatezza e quella che necessita di integrità. Facendo riferimento alla figura precedente, l'intestazione è ciò che necessita solo di integrità e autenticazione (per questo motivo chiamata anche *dati associati*), mentre il carico utile è la parte per la quale è necessaria anche la riservatezza (per questo motivo chiamata *testo in chiaro*). Questo algoritmo necessita di un'unica chiave simmetrica e di un nonce per evitare anche attacchi di replay: il risultato è il testo cifrato, che viene inserito all'interno del pacchetto come payload cifrato. Davanti a questo, in chiaro, ci sono l'intestazione e il nonce: all'interno del payload crittografato c'è una parte che garantisce l'integrità non solo del payload stesso, ma anche dell'intestazione e del nonce; di solito questa parte è chiamata **tag**.

IGE (Infinite Garble Extension)

Si tratta di un algoritmo di crittografia autenticata, che offre quindi riservatezza, integrità e autenticazione in un'unica soluzione. Fondamentalmente, è come il CBC con un'aggiunta: nel CBC, una modifica in un blocco del testo cifrato ha effetto solo sul blocco stesso e su quello successivo (e poi la propagazione si ferma). Al contrario, l'IGE utilizza un meccanismo per cui una modifica in un blocco qualsiasi influisce su tutti i blocchi successivi a quello alterato.

Questo è importante perché significa che inserendo un ultimo blocco di testo in chiaro con contenuto fisso (ad esempio, composto da tutti zeri o

contenente il numero di blocchi del testo in chiaro), nella fase di decrittazione è sufficiente guardare l'ultimo blocco e verificare se contiene il valore atteso: in caso contrario, si può dire immediatamente che c'è stato un attacco all'integrità. In questo contesto, l'ultimo blocco è quello che prima veniva chiamato **tag**.

Tuttavia, questo algoritmo non è consigliato, non è molto buono, ma il professore lo ha spiegato a scopo didattico.

Crittografia autenticata: standard

ISO/IEC 19772:2009 definisce 6 **modalità** standard (applicando queste modalità con algoritmi di crittografia di base è possibile ottenere una crittografia autenticata con i dati associati):

- **OCB 2.0 (Offset Codebook Mode)** [AEAD a passaggio singolo, brevettato];
- **AESKW (AES Key Wrap);**
- **CCM (modalità CTR con CBC-MAC)** [doppio passaggio];
- **EAX (Encrypt then Authenticate then X(trans)late) = CTR + OMAC** [AEAD a doppio passaggio, gratuito];
- **Crittografa-allora-MAC;**
- **GCM (modalità Galois/Counter).**

GCM come esempio di AEAD

Un algoritmo applicato alla crittografia in modalità GCM prende questi parametri:

- In fase di codifica, $(C, T) = \text{algoGCMenc}(K, IV, P, A)$, con:
 - Dimensione IV [1 ... 2^{64} bit] (96 bit sono i più efficienti);
 - P di dimensione [0 ... $2^{39} - 256$ bit];
 - A (dati associati autenticati) di dimensione [0 ... 2^{64} bit];
 - C ha le stesse dimensioni di P;
 - T è il tag di autenticazione con dimensione [0 ... 128 bit]: rispetto alla funzione hash crittografica, questo tag non è molto grande, e abbiamo sostenuto che un codice di integrità grande è necessario per una forte sicurezza. Si noti però che, poiché in questo caso non si utilizza un algoritmo di hash, la sicurezza è pari al numero di bit, mentre per le funzioni hash crittografiche abbiamo visto che la protezione è pari alla metà della lunghezza del digest. Quindi, in questo caso un tag di 128 bit fornisce la stessa protezione di un digest

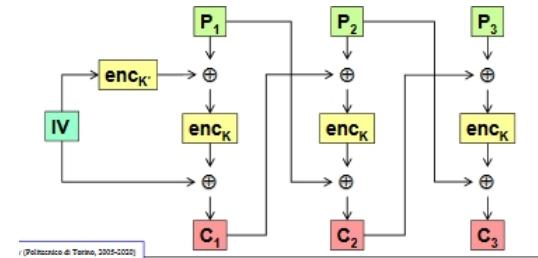


Figura 26 Funzionamento dell'Infinite Garble Extension (IGE): ogni blocco successivo a quello danneggiato sarà influenzato.

lungo 256 bit.

- Alla decrittazione, $P = algoGCMdec(K, IV, C, A, T)$, dove: P è il testo in chiaro originale (se l'autenticazione è OK), oppure un valore speciale di fallimento se i controlli di autenticazione non sono andati a buon fine.

Il GCM è definito solo per gli algoritmi di crittografia con blocco a 128 bit (ad esempio, non è possibile avere il DES nel GCM, ma è possibile avere l'AES).

CCM come esempio di AEAD

È un **Counter-mode con CBC-MAC**: prima viene calcolato un tag di autenticazione di (testo in chiaro + dati associati) da CBC-MAC, poi il testo in chiaro e il tag vengono crittografati (separatamente) in modalità CTR: si tratta quindi di un algoritmo a doppio passaggio. È definito per gli algoritmi di crittografia con blocco a 128 bit.

Crittografia autenticata: applicazioni

- TLS-1.3 utilizza GCM e CCM;
- 802.11i utilizza CCM;
- ZigBee (protocollo a corto raggio per IoT) utilizza CCM* (=CCM + auth-only + enc-only);
- ANSI C12.22 (trasmissione in rete di misure elettroniche, ad esempio contatori di energia elettrica) utilizza **EAX'** (famoso caso di **guasto**): al giorno d'oggi è comune utilizzare il cavo elettrico per inviare misure elettroniche di contatori di energia elettrica. Poiché non esiste un firewall su un cavo elettrico, attaccando un dispositivo appropriato alla rete elettrica, potrebbe essere possibile leggere le misure di altre persone o creare trasmissioni false, quindi è necessaria una protezione per la privacy (ad esempio una casa che non consuma potrebbe segnalare che non c'è nessuno in casa e quindi potrebbe essere derubata) e per l'autenticità (le misure inviate devono essere quelle reali).
 - La modifica dell'algoritmo di base ha creato una vulnerabilità, per cui è stato dimostrato che l'algoritmo è rotto ([articolo](#) di Minematsu, Morita e Iwata);
 - EAX aveva una prova formale della sua sicurezza... ma ANSI l'ha sacrificata per 3-5 passaggi di crittografia in meno e 40 byte di memoria in meno (così importante per i sistemi embedded?).

Confronto tra le modalità AE

- **GCM**: *il più diffuso, AEAD on-line a passaggio singolo, parallelizzabile, utilizzato da TLS, presente in openssl;*
 - per la crittografia, genera il testo cifrato + il tag di autenticazione;
 - per la decrittazione, calcola prima il tag di autenticazione e solo se corrisponde a quello in ingresso decifra il testo cifrato;
 - veloce su architettura Intel (~4 cicli/byte) utilizzando AES-NI per la crittografia e PCLMULQDQ per il tag.
- **OCB 2.0**: *il più veloce, AEAD on-line a passaggio singolo, brevettato GPL quindi scarsamente utilizzato, ora libero ma per usi militari;*
- **EAX**: *AEAD on-line a doppio passaggio, lento ma piccolo (utilizza solo il blocco di crittografia), quindi ottimo per i sistemi vincolati;*
- **CCM**: *doppio passaggio off-line, il più lento;*
Si noti che il doppio passaggio è due volte più lento del passaggio singolo (nel software).

Concorso per l'AE

Il tema dell'algoritmo AE è così importante che è stato indetto un concorso denominato **CAESAR** (*Competition for Authenticated Encryption: Security, Applicability, and Robustness*). La selezione finale era prevista per dicembre 2017 e avrebbe dovuto produrre un portfolio di algoritmi, ma la sfida si è conclusa senza un risultato.

Raccomandazioni NIST per l'utilizzo di algoritmi di crittografia a blocchi

Il NIST ha fornito nel corso degli anni raccomandazioni per gli algoritmi a blocchi, tutte contenute nel documento speciale

800-38, che viene aggiornato periodicamente.

- *parte A (dec'01)* = cinque modalità di riservatezza (ECB, CBC, CFB, OFB e CTR);
- *addendum (ott'10)* = tre furti di testo cifrato CBC;

- *parte B (maggio '05)* = aggiunta una modalità di autenticazione chiamata CMAC (più o meno come OMAC, che è meglio di XCBC, che è meglio di CBC-MAC);
- *parte C (luglio '07)* = aggiunta crittografia dell'autenticazione con CCM;
- *parte D (nov.'07)* = autenticazione ad alta velocità con GCM;
- *parte E (gennaio'10)* = riservatezza per le memorie a blocchi = XTS-AES;

- draft = utilizzo di AESKW per la codifica delle chiavi.

Autenticazione tramite digest e crittografia asimmetrica: firma digitale

Come abbiamo visto, la crittografia autenticata fornisce integrità, autenticazione e riservatezza ai dati, ma, come abbiamo discusso del keyed digest, non fornisce il non ripudio. Come si è visto, ciò accade perché i peer condividono la chiave su cui calcolare un keyed digest valido, quindi non è possibile dire quale dei due peer lo abbia calcolato.

Il non ripudio si ottiene utilizzando il digest e la crittografia asimmetrica: il digest viene crittografato con la chiave privata del mittente, in modo che l'uso della sua chiave pubblica provi l'origine. Utilizzando le formule, la comunicazione si svolge come segue:

- **Firmatario:** $H = enc(SK_{pri}, \text{hash}(M))$
- **Trasmissione:** $M | H$
- **Verificatore:** $X = dec(SK_{pub}, H)$
- Fase di **verifica:** se ($X == \text{hash}(M)$) allora OK altrimenti significa che qualcosa è stato modificato durante la trasmissione.

Questo tipo di schema:
non fornisce confidenzialità
fornisce non ripudio
fornisce integrità
fornisce autenticazione

Questo è il meccanismo con cui vengono implementate le **firme digitali**. L'ultimo tassello per ottenere il non ripudio completo è un meccanismo con cui deve essere possibile garantire che l'associazione tra una chiave pubblica e un'identità sia vera: si tratta del cosiddetto **certificato a chiave pubblica**.

Un attacco che è possibile eseguire contro questo schema è la *modifica della firma digitale*: la modifica di qualsiasi bit dei dati crittografati porta al fallimento della fase di verifica, ma anche qualsiasi modifica alla firma digitale: non importa se i dati sono corretti, non è possibile dire se la modifica ha interessato i dati o la firma.

Un altro caso che merita di essere discusso è quello di prendere una firma digitale eseguita da qualcun altro, poi prendere anche la chiave pubblica di quella persona (per assicurarsi che la chiave pubblica corrisponda alla firma digitale), e poi allegarla a un altro file, in modo che venga attribuita a quella persona: questo comporterà comunque un fallimento nella fase di verifica, poiché la firma digitale contiene il digest crittografato dei dati originali: la firma digitale è legata ai dati, quindi non è possibile *allegare una firma ad altri dati*.

Autenticazione e integrità: analisi

Abbiamo visto due modi per ottenere l'autenticazione e l'integrità:

- per mezzo di un **segreto condiviso**:
 - utile solo per il ricevente;
 - non può essere utilizzato come prova senza rivelare la chiave segreta;
 - non è utile per il non ripudio.
 - Infine, è utile per due persone che si fidano l'una dell'altra, per proteggersi da una terza persona;
- mediante **crittografia asimmetrica**:
 - essendo lento, viene applicato solo al digest;
 - può essere utilizzato come prova formale;
 - può essere utilizzato per il non ripudio, a condizione che sia possibile dimostrare chi è il

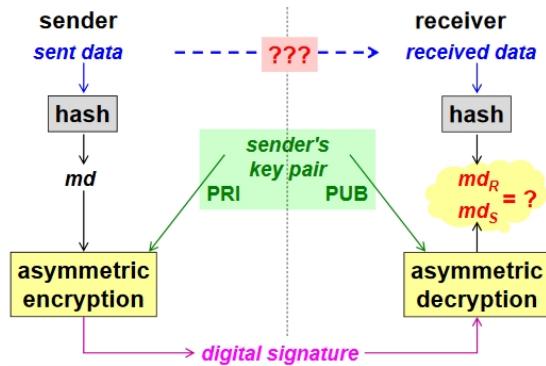


Figura 27 Firma digitale: calcolo e verifica del digest di un messaggio crittografato tramite crittografia asimmetrica

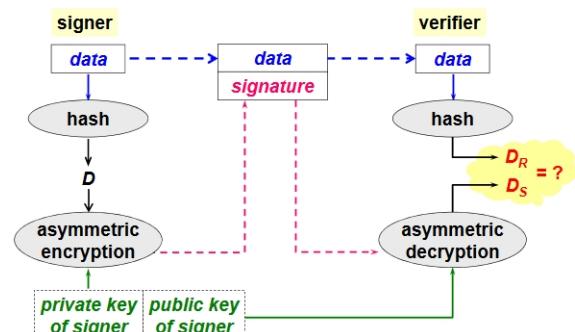


Figura 28 Creazione e verifica della firma

proprietario della chiave pubblica ([certificato a chiave pubblica](#));

Firma digitale e firma autografa

La firma digitale fornisce autenticazione e integrità, mentre la firma autografa fornisce solo l'autenticazione. Pertanto, la firma digitale è migliore perché è strettamente legata ai dati. Nota bene: ogni utente non possiede una firma digitale ma una chiave privata, che può essere utilizzata per generare un numero infinito di firme digitali (una per ogni diverso documento).

Certificato a chiave pubblica

Come abbiamo visto finora, la firma digitale ha un grande vantaggio, ovvero può fornire il non ripudio. Tuttavia, il non ripudio si ottiene se, oltre a eseguire una firma digitale corretta, è possibile attribuire con certezza la chiave pubblica alla chiave privata di un attore. L'attribuzione di una chiave a un attore avviene solitamente attraverso un **certificato di chiave pubblica**: si tratta di "una struttura di dati utilizzata per legare in modo sicuro una chiave pubblica ad alcuni attributi". In genere, lega una chiave a un'identità, ma sono possibili anche altre associazioni (ad esempio, l'indirizzo IP). La sicurezza di questo legame è garantita da una firma digitale: l'entità che ha creato il certificato, chiamata **autorità di certificazione (CA)**, firma digitalmente il certificato, in modo che sia allo stesso tempo:

- una prova di *autenticazione*: è un certificato valido perché è stato creato da un'autorità di certificazione;
- una prova di *integrità*: i dati sono corretti e non sono stati modificati dopo l'emissione del documento.

Come un comune documento di identità, anche un certificato a chiave pubblica ha una **durata limitata**. Inoltre, può essere revocato su richiesta sia dall'utente che dall'emittente.

Formati per i certificati a chiave pubblica

- **X.509**:
 - v1, v2 (definiti dall'ISO): non hanno avuto molto successo;
 - v3 (ISO + IETF): ha raggiunto un grande utilizzo, è oggi il formato più comune, perché può essere applicato alla sicurezza di diversi protocolli Internet;
 - **non X.509**: poiché X.509 è stato sviluppato dall'ISO, che di solito è visto come l'ente di standardizzazione delle grandi aziende, ci sono persone che non si fidano delle grandi aziende e si sono impegnate a creare strutture di certificazione alternative, come ad esempio:
 - **PGP**: è stato usato (e viene tuttora usato) soprattutto nel movimento clandestino;
 - **SPKI** (IETF)
- Nessuno di questi ha raggiunto una grande diffusione.
- **PKCS#6**: è uno standard promosso dalla RSA prima dell'invenzione di X.509, quindi è parzialmente compatibile con X.509, ma da allora la stessa RSA lo ha dichiarato obsoleto.

Struttura di un certificato X.509

Un certificato X.509 è composto da:

- **Versione**: poiché esistono più versioni dello stesso formato, come visto in precedenza;
- **Numero di serie**: ogni certificato è identificato in modo univoco;
- **Algoritmo di firma**: poiché il certificato è protetto da una firma digitale, nel suo formato sono presenti le informazioni sull'algoritmo utilizzato per eseguire la firma;
- **Emittente**: è il nome dell'entità che ha creato il certificato, l'autorità di certificazione. La sintassi utilizzata per specificare queste informazioni è chiamata **DN** (distinguished name), composta da tre campi diversi:
 - **C** sta per paese (ad esempio, Italia);
 - **O** sta per organizzazione (es. Politecnico di Torino);

2
1231
RSA with MD5, 1024
C=IT, O=Polito, OU=CA
1/1/97 - 31/12/97
C=IT, O=Polito, CN=Antonio Lioy Email=lioy@polito.it
RSA, 1024, xx...x
yy...y

Figura 29 Esempio di struttura di un certificato a chiave pubblica X.509

- **OU** sta per organizational unit (unità organizzativa), il che significa che si tratta del dipartimento che ha creato il certificato (ad esempio, l'autorità di certificazione).
- **Validità:** definisce il periodo per il quale il certificato è considerato valido. Al di fuori di tale periodo, la CA non garantisce la corrispondenza.
- **Soggetto:** l'entità che controlla la chiave privata corrispondente alla chiave pubblica da certificare. Inoltre, qui viene utilizzato il DM per identificare tale entità, con altri campi non menzionati in precedenza:
 - **CN** sta per nome comune (ad esempio, Antonio Lioy);
 - **Email**, ovvero l'email dell'entità. È importante includere questo campo perché i certificati a chiave pubblica sono utilizzati per proteggere le applicazioni Internet, ad esempio per inviare un'e-mail ad "Antonio Lioy", il server di posta deve essere in grado di capire qual è la posta certificata dell'entità (quindi, il campo e-mail è obbligatorio). Lo stesso ragionamento vale per altre applicazioni per le quali si desidera una protezione.
- **subjectPublicKeyInfo:** è il campo contenente la chiave pubblica; specifica l'algoritmo, la lunghezza della chiave e tutti i bit che la costituiscono;
- **Firma digitale della CA:** è la firma digitale dell'autorità di certificazione calcolata sul certificato; in questo modo è possibile rilevare qualsiasi modifica ad esso, come visto [in precedenza](#).

Quindi, questo meccanismo risolve finalmente il problema di sapere chi è l'attore che controlla la chiave privata corrispondente alla chiave pubblica.

PKI (Infrastruttura a chiave pubblica)

Si tratta di un'infrastruttura che svolge due tipi di compiti: *tecnici* (creazione di certificati) e *amministrativi* (prima di creare un certificato, è necessario verificare l'identità del richiedente). Quindi, questa infrastruttura supervisiona la *creazione*, la *distribuzione* e la *revoca* dei certificati a chiave pubblica.

Revoca del certificato

Ogni certificato ha una certa validità, ma può essere revocato prima della data di scadenza. La revoca può essere richiesta dal proprietario o dal creatore del certificato:

- Il proprietario può richiedere la revoca nel caso in cui non abbia più il controllo sulla chiave privata (ad esempio, una chiave privata memorizzata su una smartcard rubata). Questo caso è simile a quello dello smarrimento di una carta di credito: il proprietario dovrebbe chiedere la revoca del certificato il prima possibile, perché se il ladro usa quella chiave privata, sembrerà che il proprietario originale abbia eseguito la firma. Questo è il caso più comune in cui un certificato viene revocato;
- Il creatore può revocare il certificato nel caso in cui sia stato creato per un'identità falsa: infatti, se qualcuno è riuscito a dimostrare un'identità falsa a un'autorità di certificazione, riceverà un certificato valido. Tuttavia, se in seguito l'autorità di certificazione scopre l'inganno, revoca automaticamente il certificato.

Quando un certificato viene revocato, l'autorità di certificazione crea un elenco che indica che il certificato non è più valido. L'autorità di certificazione non sa quando e dove verrà utilizzata la chiave rubata, quindi quando una firma viene utilizzata per proteggere alcune transazioni, la transazione viene ricevuta da alcuni attori (i ricevitori) che devono verificare che il certificato fosse valido al momento della firma. Questo attore (che riceve la firma) e ha il compito di verificare se la firma è valida o meno è chiamato **relying party (RP)**.

Ciò significa che prima di fidarsi del certificato, occorre verificare se è valido o meno.

Meccanismi di revoca

Esistono due meccanismi per effettuare un controllo sulla firma:

- **CRL (Certificate Revocation List):** l'autorità di certificazione crea un elenco di certificati revocati. L'elenco è firmato digitalmente dalla CA, perché altrimenti qualcuno potrebbe cancellare un certificato dall'elenco o viceversa aggiungervi un certificato per "bloccare" qualcuno: la CA garantisce l'autenticità e l'integrità di questo elenco firmandolo digitalmente. Poiché si tratta di un

elenco con **tutti i** certificati revocati, questo

permettono al destinatario di verificare la validità dal momento dell'emissione del certificato. Ad esempio, se oggi si riceve un documento che è stato firmato 3 mesi fa, occorre verificare se la chiave era valida 3 mesi fa. Se il certificato è stato rimosso oggi, questo non influisce sul documento ricevuto, perché il controllo deve essere fatto rispetto al momento della firma. Il problema è che dobbiamo scaricare l'elenco completo solo per controllare un singolo certificato, ma questo è l'unico modo per conoscere la storia di un certificato.

- **OCSP (On-line Certificate Status Protocol):** questo meccanismo è preferito per i controlli in tempo reale, ad esempio per le transazioni. In questo caso, è necessario verificare se il certificato è valido in questo momento e il servizio che necessita del controllo non è interessato agli stati precedenti del certificato. OCSP è un protocollo client-server in cui è possibile richiedere a un servizio specifico se il certificato è valido o meno al momento attuale. La risposta è firmata dal server, in modo che non sia possibile fornire una risposta falsa.

Struttura di una CRL X.509

Certificate Revocation List (CRL)

La CRL X.509 ha gli stessi parametri del certificato X.509, quali: versione, algoritmo di firma ed emittente. Ma la struttura della CRL X.509 ha il parametro "thisUpdate", che è l'ora in cui la CRL è stata creata. Questo parametro è importante perché se si esegue un controllo su una firma realizzata prima di questa data, l'informazione è contenuta nella CRL. Ma se il controllo deve essere effettuato su qualcosa avvenuto dopo la data di questo aggiornamento, allora questa CRL non va bene per i nostri scopi e dobbiamo ottenere una CRL più fresca. Dopo questo parametro, c'è un elenco dei certificati revocati: per ogni certificato sono riportati l'userCertificate e la revocationDate.

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

Verifica di una firma/certificato

Supponiamo di voler verificare una firma: la verificheremo richiedendo il certificato della chiave pubblica, che è firmato dall'entità che lo fornisce (ad esempio, il certificato di "Antonio Lioy" è firmato dal "Politecnico di Torino"). A questo punto sorge una domanda: come è possibile verificare che quest'ultima firma sia valida? Per verificare se la firma della CA è valida, è necessario un certificato a chiave pubblica di quella CA, che viene creato e firmato da un'altra CA, la cui firma a sua volta deve essere

verificata. È facile intuire che il problema è ricorsivo: diventa quindi necessario disporre di un'infrastruttura gerarchica per la certificazione e la distribuzione dei certificati a chiave pubblica.

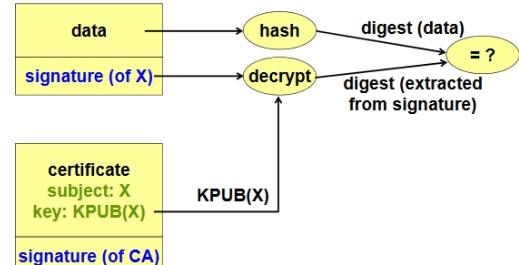


Figura 30 Il problema che si presenta quando si verifica una firma su dati inviati da un attore. Per verificare la firma sui "dati", è necessario il certificato di chiave pubblica di X, fornito da una CA. A sua volta, questo certificato necessita dello stesso trattamento: la firma della CA deve essere convalidata.

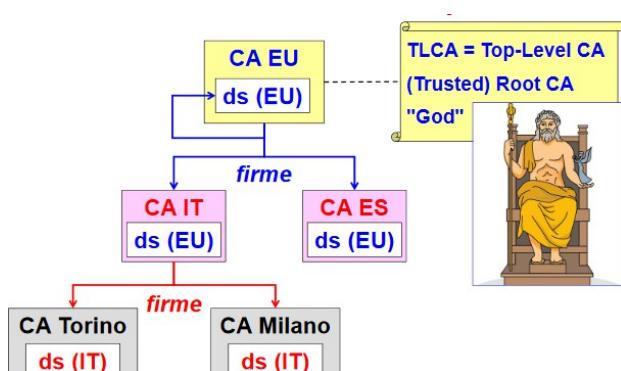


Figura 31 L'infrastruttura gerarchica per la certificazione e la distribuzione dei certificati a chiave pubblica.

Questa infrastruttura gerarchica è rappresentata a sinistra. Ad esempio, per verificare la firma del "Politecnico di Torino", che ha creato il certificato di chiave pubblica per "Antonio Lioy", è necessario il certificato della "CA Torino", che contiene la firma della "CA IT"; per verificare se questa è buona, è necessario il certificato dell'entità che ha firmato il certificato della "CA IT", quindi la "CA EU". Quest'ultima CA conclude la catena, infatti la "CA EU" è firmata dalla stessa CA europea: quest'ultimo certificato è fidato a priori (autofirmato), non può essere verificato, e per questo motivo è chiamato

anche TLCA, **Root CA** o addirittura **Dio**.

Si noti che al mondo non esiste un'unica CA fidata, né ce ne sono poche: in genere, nei sistemi informatici ci sono molte CA fidate, e questo è dovuto a due ragioni principali:

- Sicurezza:** chi controllasse l'unico TLCA controllerebbe anche gli altri, falsificandoli;
- Commerciale:** per ottenere un certificato da un'autorità di certificazione, è necessario pagare un prezzo.

Per il secondo motivo, l'elenco delle CA fidate può essere diverso nei sistemi prodotti da produttori diversi: poiché è una questione di chi pone la fiducia, un sistema può accettare o meno una CA come fidata. Ad esempio, in passato Microsoft effettuava solo lievi controlli sulle CA fidate in entrata, ma richiedeva il pagamento di 250.000 dollari per entrare nel suo elenco.

Il problema che si pone è: un aggressore non può falsificare l'intero albero? Naturalmente è possibile: se il dispositivo viene lasciato incustodito e l'attaccante riesce ad avere privilegi amministrativi, è possibile aggiungere un'altra Root CA (falsa), rendendo l'intero sistema gerarchico inutilizzabile per quel dispositivo. L'elenco delle CA radice può essere gestito dal sistema operativo o dall'applicazione utilizzata.

Prestazioni

Le prestazioni crittografiche non dipendono dalla RAM ma dalla CPU (architettura e set di istruzioni) e dalla dimensione della cache: alcune CPU hanno istruzioni speciali per gli algoritmi crittografici. In generale, le prestazioni non sono un problema sui client (tranne quando non sono molto potenti, ad esempio nel caso di IoT, sistemi embedded o client sovraccaricati da altre applicazioni). Le prestazioni possono diventare un problema sui server o quando si implementa la sicurezza sui nodi di rete (ad esempio, router); in questi casi, è possibile utilizzare:

- **acceleratori crittografici (HSM, Hardware Security Module):** dispongono di alcune istruzioni primitive per eseguire alcuni algoritmi di sicurezza;
- **acceleratori per scopi speciali,** cioè acceleratori che non solo implementano in hardware le operazioni crittografiche ma anche l'elaborazione dei pacchetti per alcuni protocolli di sicurezza (ad esempio, SSL, IPsec), o **acceleratori generici**, che ad esempio si limitano a velocizzare il calcolo di AES per qualsiasi protocollo si voglia implementare.

Le figure seguenti mostrano esempi di prestazioni: è possibile notare che per il calcolo di funzioni hash crittografiche, le prestazioni sono molto migliori se l'operazione coinvolge pacchetti di dimensioni maggiori. Questo non è più vero quando si parla di algoritmi di crittografia, ma è possibile notare che, ovviamente, 3DES è tre volte più lento del normale DES, che AES è più veloce

del DES anche se garantisce una sicurezza molto migliore, e che l'RC4 è molto più veloce degli altri perché è un algoritmo di flusso.

	[64 B/packet]	[1024 B/packet]
hmac(md5)	193.8 MB/s	630.5 MB/s
des-cbc	90.5 MB/s	91.0 MB/s
des-ede3-cbc	33.7 MB/s	33.9 MB/s
aes-128-cbc	171.3 MB/s	178.4 MB/s
rc4	783.1 MB/s	637.9 MB/s

Figura 32 Prestazioni delle funzioni hash e degli algoritmi di crittografia su una CPU potente (Intel i7-7600U @ 2,80GHz)

	[64 B/packet]	[1024 B/packet]
hmac(md5)	19.2 MB/s	83.6 MB/s
des cbc	14.4 MB/s	14.5 MB/s
des ede3	5.2 MB/s	5.2 MB/s
aes-128	15.6 MB/s	15.9 MB/s
rc4	80.9 MB/s	86.4 MB/s

Figura 34 Prestazioni delle funzioni hash crittografiche e degli algoritmi di crittografia su una CPU utilizzata nei sistemi embedded (Intel P3 @ 800 MHz)

un fattore dieci: l'aumento della lunghezza della chiave nella crittografia asimmetrica porta a una riduzione esponenziale delle prestazioni.

Considerando la crittografia asimmetrica, le prestazioni non sono solitamente espresse in MB/s, ma dal numero di firme create o convalidate in un secondo. È possibile notare che questi algoritmi sono normalmente orientati verso la verifica della firma, che è più veloce: il motivo è che normalmente una firma viene creata una volta sola, ma forse deve essere verificata molte volte. Si noti anche che raddoppiando la dimensione della chiave non si dimezza il tempo di verifica della firma.

rsa 1024	12093.7 firme/s	192952.5 verifiche/s
rsa 2048	1801.4 firme/s	64765.0 verifiche/s

Figura 33 Prestazioni degli algoritmi di crittografia asimmetrica su una CPU potente (i7-7600U @ 2,80GHz)

rsa 1024 94.8 signs/s 1682.0 verifies/s

Figura 35 Prestazioni degli algoritmi di crittografia asimmetrica su una CPU utilizzata nei sistemi embedded (Intel P3 @ 800 MHz)

Nei sistemi embedded di solito si usano CPU vecchie e questo può portare a prestazioni molto scarse rispetto a quelle di una CPU moderna. Quindi, in generale, prima di decidere quale algoritmo utilizzare, si dovrebbe effettuare un test in tempo reale di tale algoritmo.

algoritmo sulla piattaforma hardware specifica che verrà utilizzata per l'applicazione di destinazione.

Soluzioni suggerite per la sicurezza

NSA suite B (2005-2017) e CNSA (2018)

Abbiamo visto diversi algoritmi, diverse modalità di applicazione e in effetti esistono alcune linee guida su cosa utilizzare. Una di queste è quella della **NSA** (National Security Agency of United States) che ha raccomandato la **suite B** nel periodo 2005-2017: il nome "suite B" deriva dal fatto che la "suite A" è l'insieme di algoritmi che solo la NSA utilizza e non divulga. Quindi, la suite B è quella che l'NSA suggerisce al governo degli Stati Uniti di utilizzare quando acquista prodotti COTS (Commercial Off-The-Shelf) che trattano informazioni SBU (Sensitive But Unclassified) e classificate. Contiene i seguenti algoritmi:

- **(crittografia simmetrica)** AES-128 e AES-256;
- **(hash)** SHA-256 e SHA-384;
- **(accordo chiave)** ECDH e ECMQV;
- **(firma digitale)** ECDSA.

Per le informazioni fino al livello Secret suggerisce AES-128 + SHA-256 + EC P-256; per le informazioni a livello Top Secret suggerisce AES-256 + SHA-384 + EC P-384: gli algoritmi sono gli stessi ma la chiave o il digest è più lungo.

Nel 2018 la NSA ha rilasciato una nuova suite denominata **CNSA** (Commercial National Security Algorithm Suite). Essa comprende i seguenti algoritmi:

- **(crittografia simmetrica)** AES-256: ora qualsiasi chiave più corta di 256 bit non è considerata sufficiente;
 - Per proteggere i dati in tempo reale (ad esempio, i flussi), suggerisce la modalità CTR per le applicazioni a bassa larghezza di banda o GCM per quelle che richiedono una larghezza di banda elevata. Inoltre, GCM fornisce anche una crittografia autenticata, il che rappresenta un ulteriore vantaggio;
- **(hash)** SHA-384;
- **(accordo chiave)** ECDH e ECMQV;
- **(firma digitale)** ECDSA, con EC sulla curva P-384.

Per i sistemi legacy, ad esempio i vecchi sistemi ancora funzionanti che non possono eseguire EC o GCM, si suggerisce di utilizzare:

- **(accordo chiave)** DH-3072;
- **(scambio di chiavi, firma digitale)** RSA-3072;
- nuovi algoritmi resistenti ai quanti previsti entro il 2022.

Lunghezza delle chiavi e digest (NIST, 2011)

Nel 2011 il NIST ha pubblicato una raccomandazione che indica la lunghezza suggerita delle chiavi in base al numero di anni di protezione. Nella tabella seguente:

- **FFC** è l'acronimo di Finite Field Cryptography (ad esempio, DSA, DH);
- **IFC** sta per Integer Factorization Cryptography (ad esempio, RSA).

symm.	FFC	IFC	ECC	hash	years
80	1024	1024	160	160	< 2010
112	2048	2048	224	224	< 2030
128	3072	3072	256	256	> 2030
192	7680	7680	384	384	>> 2030
256	15360	15360	512	512	>>> 2030

Figura 36 Relazione tra lunghezza della chiave e necessità di protezione in termini di anni per diversi tipi di algoritmi

È possibile notare che la lunghezza della chiave deve essere molto lunga se i dati criptati ora devono rimanere tali per anni dopo il 2030: il requisito delle chiavi per le ultime due righe rende l'algoritmo completamente irrealizzabile data la velocità delle CPU attuali.

Problemi di sicurezza nei prodotti importati: una retrospettiva

Oggi ci sono alcuni motivi per dubitare di prodotti che contengono in qualche modo meccanismi di sicurezza forniti da altre nazioni: ad esempio, si discute se utilizzare o meno le apparecchiature 5G.

prodotti da Huawei, perché si teme che Huawei possa inserire malware o software di spionaggio all'interno del proprio prodotto. Questa situazione non è nuova, infatti ci sono già stati precedenti di questo tipo.

Dal 1990 negli Stati Uniti l'esportazione di materiale crittografico era soggetta alle stesse restrizioni del materiale nucleare, a meno che il livello di protezione non fosse molto basso: Le aziende americane che volevano esportare il loro software contenente soluzioni di sicurezza furono costrette ad abbassare la protezione, limitando la lunghezza della chiave simmetrica a 40 bit e quella asimmetrica a 512 bit (anche a quei tempi, il calcolo per infrangere una tale protezione avrebbe richiesto solo poche ore di CPU). Esempi di tali limitazioni sul software sono il caso delle versioni esportate di Netscape e Internet Explorer: questi software erano ostacolati, quindi avevano una forza ridotta rispetto alla versione statunitense.

Ciò ha spinto alcune aziende di tutto il mondo a sviluppare soluzioni proprie, cosicché le aziende americane stavano perdendo mercato: dopo aver protestato contro il governo, nel dicembre 1996 il governo statunitense ha autorizzato l'esportazione di prodotti crittografici semi-robusti (56 bit) se incorporano funzioni di **key-escrow**. Il **key escrow** è la possibilità di recuperare una chiave anche senza il consenso del proprietario. La maggior parte delle aziende si rifiutò di implementare questa soluzione, ma ci fu il caso notevole del famoso software *Lotus Notes 4.x*, che utilizzava chiavi simmetriche a 64 bit, ma 24 bit di queste erano criptate con la chiave pubblica dell'NSA: in questo modo l'NSA poteva facilmente recuperare questi 24 bit e calcolare i restanti 40 bit, pur avendo la protezione standard per chiunque altro. Il problema è che non c'è alcuna dichiarazione su chi decide quando è necessario recuperare una chiave.

Le numerose edizioni di Lotus Notes

"Oltre al tempo del processo di crittografia, le leggi governative statunitensi sulle esportazioni limitano la lunghezza delle chiavi di crittografia. Queste leggi sono la forza trainante delle [tre edizioni principali di Notes: Nord America, Internazionale e Francese](#). Nonostante i nomi diversi, la funzionalità del prodotto è esattamente la stessa. La differenza, tuttavia, sta nella lunghezza delle chiavi utilizzate per la crittografia. L'edizione nordamericana utilizza chiavi di crittografia a 64 bit. Il governo degli Stati Uniti, per motivi di sicurezza nazionale, limita la lunghezza delle chiavi di crittografia per l'esportazione a 40 bit. Per rispettare queste restrizioni, abbiamo l'edizione internazionale.

Quando generiamo una chiave a 64 bit per l'edizione internazionale, i primi 24 bit vengono crittografati utilizzando la chiave pubblica del governo degli Stati Uniti e memorizzati nel cosiddetto campo di riduzione del fattore di lavoro (WRF). Dividendo la chiave in questo modo si ottiene una chiave di 40 bit per il Governo degli Stati Uniti e di 64 bit per tutti gli altri. Questo approccio consente di mantenere un elevato livello di sicurezza a livello mondiale senza violare le leggi sull'esportazione del governo degli Stati Uniti.

La maggior parte dei Paesi è soddisfatta del modo in cui l'edizione internazionale è conforme alle leggi statunitensi sull'esportazione delle chiavi di crittografia. **Il governo francese, tuttavia, ha ritenuto inaccettabile l'edizione internazionale. Per rispettare la legge francese, abbiamo creato l'edizione francese, che utilizza una chiave di crittografia semplice a 40 bit** e può quindi essere "rotta" da aggressori disposti ad applicare una notevole potenza di calcolo (presumibilmente, compreso il governo francese)".

Modifiche alle normative statunitensi sull'esportazione di crittografia

Casi come questo hanno continuato a favorire la creazione di altre soluzioni, così le normative USA sono cambiate:

- **Giugno 1997:** l'autorizzazione all'esportazione di client e server web sicuri è stata concessa solo se utilizzati da filiali estere di aziende statunitensi o in ambito finanziario (transazioni). Per verificare l'effettivo utilizzo, è necessario utilizzare speciali certificati emessi da Verisign, che utilizzano un meccanismo chiamato **crittografia step-up o gated**;
- **Settembre 1998:** autorizzazione estesa alle istituzioni assicurative e sanitarie e nessuna autorizzazione per chiavi fino a 56 bit (perché all'epoca era chiaro che 56 bit non erano sufficienti).

Crittografia step-up (gated) (dec'98)



Supponiamo di essere una banca (lato destro dell'immagine) che può acquistare software statunitense e il server web conterrà una crittografia forte, ma l'utente è italiano con un browser italiano che ha solo una crittografia debole. Quando entrambi devono comunicare insieme, l'unica soluzione è utilizzare la crittografia più debole disponibile. Per questo motivo, le banche hanno chiesto al governo

come rendere possibile la comunicazione ed ecco il trucco: la versione internazionale del programma contiene anche la parte *crittografica forte*, che è bloccata e non può essere utilizzata. Ma se una banca acquista il software, può anche acquistare un *certificato pubblico* da Verisign (l'unica CA di cui il governo degli Stati Uniti si fida) che può verificare che il richiedente sia davvero una banca e non un falso. Una volta che la banca ha il certificato Verisign, può inviare come parte della transazione anche il certificato. Il certificato apre magicamente la porta della crittografia forte, ma *solo* per la comunicazione con la banca. Questa è la cosiddetta **crittografia a gradini**, perché si parte dalla debolezza e si fa un passo avanti con una crittografia più forte. Si chiama anche **crittografia a cancello** perché la parte forte è all'interno di un cancello che si apre solo quando il server presenta il certificato corretto.

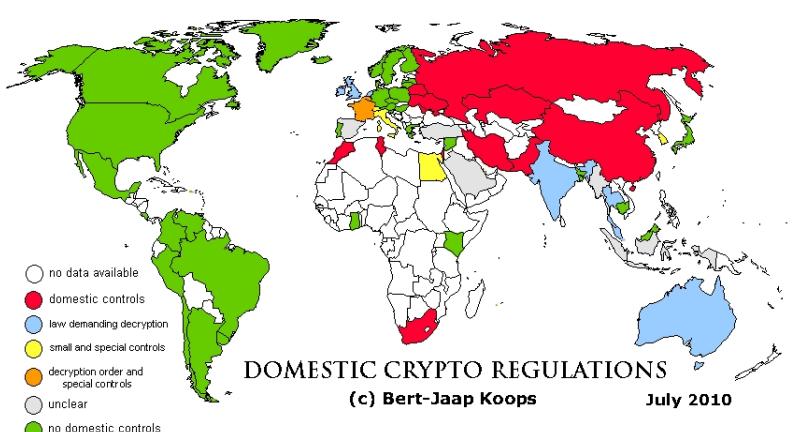
Recupero delle chiavi: un'altra novità?

In seguito il governo ha introdotto un'altra regola: è possibile esportare prodotti con crittografia forte (ad esempio, chiavi simmetriche a 128 bit) se sono incorporate le **funzioni di recupero delle chiavi**. Il recupero della chiave significa che se qualcosa viene crittografato e poi la chiave viene persa, è possibile avere un recupero della chiave per leggere ancora i dati crittografati. Queste chiavi, create insieme alla crittografia, devono essere collocate in un **centro di recupero** autorizzato dal governo statunitense. Purtroppo, tutti questi centri sono stati collocati all'interno di basi militari statunitensi (il che non è molto affidabile). In questo modo tutte le questioni diventano problemi politici: ma se siete un'azienda e avete deciso di proteggere i dati con la crittografia, dovete sapere bene cosa succede se la chiave viene persa, perché se ciò accade potrebbero esserci problemi davvero enormi.

Con il nuovo millennio (Gen '00) è stata introdotta una nuova normativa che autorizza l'esportazione di prodotti off-the-shelf che hanno superato una "one-time review" o di prodotti il cui codice sorgente è liberamente disponibile su Internet.

Sintesi

Questo è il fattore comune a quanto detto sopra riguardo alle leggi statunitensi e alle recenti preoccupazioni sull'importazione di hardware e software 5G: tutti i governi cercano di spiare i cittadini, e potrebbe essere per buone ragioni, ma questo dà anche preoccupazioni sulla libertà, infatti un governo potrebbe spiare i cittadini per capire se sono contro il governo stesso. Oggi, allo stesso modo in cui gli Stati Uniti hanno fatto in passato, ci sono alcuni dubbi che all'interno dell'apparecchiatura che produttori come Huawei



produrre uno spyware o un malware controllato dal governo cinese.

Figura 37 Questo è un riepilogo grafico delle leggi e dei regolamenti in materia di crittografia in tutto il mondo, come indicato nella versione più recente del [Crypto Law Survey](#).

Tecniche e architetture di autenticazione

Esistono diverse definizioni di *autenticazione*:

- RFC-4949 (**glossario della sicurezza di Internet**):
 - "il processo di verifica di un'affermazione secondo cui un'entità di sistema o una risorsa di sistema possiede un determinato valore di attributo";
- **whatism.com**:
 - "il processo per determinare se qualcuno o qualcosa è chi o cosa si dichiara di essere";
- NIST IR 7298 (**Glossario dei termini chiave della sicurezza informatica**):
 - "verifica dell'identità di un utente, di un processo o di un dispositivo, spesso come prerequisito per consentire l'accesso alle risorse di un sistema informativo".

I punti importanti di queste definizioni sono che definiscono l'autenticazione di un attore, il che significa che potrebbe essere non solo un essere umano (che interagisce tramite un software in esecuzione su hardware), ma anche un componente software o un elemento hardware (che interagisce tramite software). L'abbreviazione comune per l'autenticazione è *authN* o *authC*, mentre *authZ* è usato per l'autorizzazione, che è diversa ma correlata.

Per l'autenticazione di un attore si possono utilizzare tre categorie di **fattori di autenticazione**:

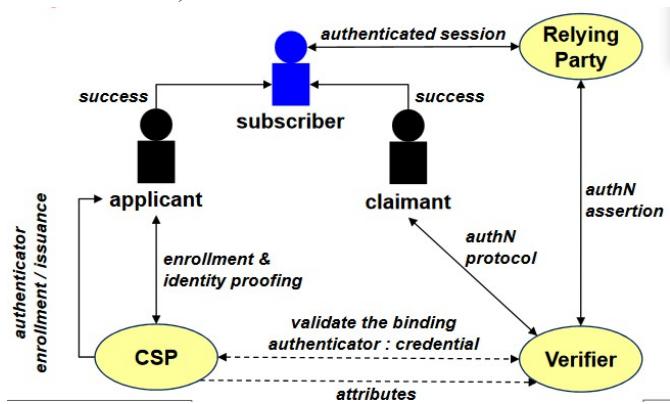
- **Conoscenza**: l'autenticazione si basa su qualcosa che *solo l'utente conosce*, ad esempio una passphrase statica, un codice, un numero di identificazione personale. Il rischio associato risiede nella memorizzazione, nel modo in cui è possibile dimostrare tale conoscenza e nel modo in cui viene trasmessa;
- **Proprietà**: l'autenticazione si basa su qualcosa che *solo l'utente possiede* (spesso chiamato "autenticatore"), ad esempio un token, una smart card, uno smartphone. I rischi associati possono risiedere nell'autenticatore stesso: può essere infettato da un malware, oppure può essere prodotto in un Paese che impone un controllo governativo su di esso, oppure può essere rubato, clonato o utilizzato senza l'autorizzazione del proprietario (ad esempio, dimenticando uno smartphone sbloccato);
- **Inerenza**: *qualcosa che l'utente è*, ad esempio una caratteristica biometrica (come l'impronta digitale). I rischi associati possono riguardare la contraffazione e la privacy: è molto peggio dei casi precedenti, perché ad esempio una caratteristica biometrica non può essere sostituita se "compromessa". Per questo motivo, i fattori di ereditarietà dovrebbero essere limitati ad ambienti molto sicuri, in genere dovrebbero essere utilizzati solo per l'autenticazione locale, come meccanismo per sbloccare un segreto o un dispositivo.

Modello di autenticazione digitale (NIST SP800.63B)

In questo modello, un attore che vuole utilizzare un sistema è chiamato *richiedente*: se possiede un autenticatore può fornirlo al **CSP** (Credential Service Provider), oppure può ottenerlo (ad esempio, quando uno studente si iscrive al Politecnico, gli viene consegnata una smart card che funziona da autenticatore). Il **CSP** è quel componente che emette o registra le credenziali e l'autenticatore dell'utente e verifica e memorizza gli attributi associati.

Quando questa procedura viene completata con successo, l'attore diventa un *sottoscrittore*, cioè un'entità

registrato nel sistema di autenticazione. In seguito,



quando l'attore vuole utilizzare un servizio di rete,

In genere l'attore è chiamato *claimant*, perché afferma di essere un utente valido: tipicamente, viene eseguito un protocollo di autenticazione contro un **verificatore**, che verifica questa affermazione. Quando questo processo termina con successo, l'attore diventa un sottoscrittore con una sessione autenticata aperta con il **relying party**, che richiederà e riceverà un'asserzione authN dal verificatore per valutare l'identità (e gli attributi) dell'utente. Il relying party è l'applicazione finale

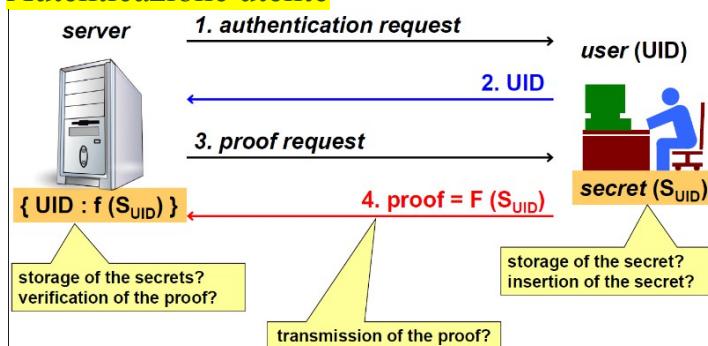
server, che richiede all'attore di essere autenticato. Il verificatore può comunicare con il CSP per convalidare il legame tra l'autenticatore utilizzato nel protocollo di autenticazione e la credenziale richiesta.

La credenziale lega un autenticatore al sottoscritto, tramite un ID: ad esempio, un certificato X.509 può essere considerato la credenziale, perché lega l'identità e gli attributi che sono scritti all'interno del certificato con l'autenticatore, che in questo caso è la chiave privata che l'utente controlla.

Questi ruoli possono essere separati o riuniti insieme. Pensando a una macchina Linux utilizzata localmente: la fase di enrollment consiste nella creazione di un nuovo utente con nome utente, che è la credenziale, e password, che è l'autenticatore. In questo scenario il CSP è il sistema operativo stesso e quando un utente vuole utilizzare un server di questa macchina, deve eseguire il login, che è il verificatore. Il relying party, infine, è qualsiasi software in esecuzione su quella macchina che utilizza l'identità dimostrata dal servizio di login del sistema operativo.

Un altro esempio è l'uso di Google Identity per diversi servizi, come il servizio Doodle per concordare una data. Per Doodle è possibile utilizzare le credenziali di Google o di Facebook. In questo caso la Relying Party è Doodle, mentre il Verificatore (così come il CSP) è Google o Facebook.

Autenticazione utente



L'utente vuole accedere a un server applicativo (Relying Party). In questo caso il server comprende sia la Relying Party che il Verifier. L'utente è stato identificato con l'ID utente e ha un segreto associato a tale ID utente. L'identificatore del server contiene una tabella con l'ID utente e il risultato della funzione f sul segreto.

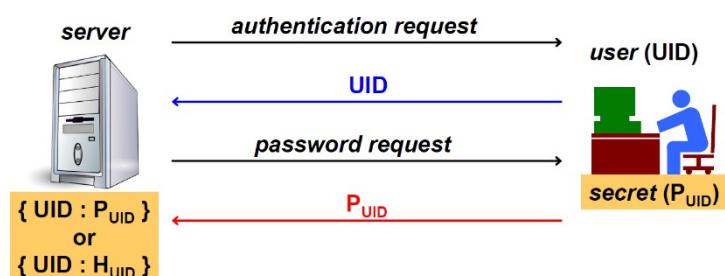
Normalmente il segreto non dovrebbe mai essere memorizzato in memoria, ma ovviamente se la funzione f è l'identità, significa che si sta memorizzando il segreto in memoria.

in chiaro, il che non è consigliabile.

Quando l'utente vuole accedere al servizio, riceve una *richiesta di autenticazione*, fornirà prima l'*UID* e poi il verificatore chiederà una *richiesta di prova*, e l'utente risponderà con la prova, che è il risultato del calcolo con la funzione F sul segreto dell'utente. In questo schema ci sono diversi problemi da affrontare: *dal lato dell'utente*, come viene memorizzato il segreto? Come viene fornito il segreto (ad esempio, se si tratta di una password inserita tramite tastiera, un keylogger potrebbe rivelarlo)? La trasmissione della prova è sicura? E *sul lato server*, come vengono memorizzati i segreti relativi all'utente? Quando si riceve una prova, come si verifica che sia quella corretta?

Password (riutilizzabile)

Immaginiamo che il segreto sia una **password riutilizzabile** (cioè sempre la stessa), che l'utente sia identificato dal suo ID utente e che il segreto sia la password associata a quell'utente. Lato server, c'è una tabella che contiene nomi utente e password in chiaro o una funzione H calcolata sulla password. Anche in questo caso, ci sarà una *richiesta di autenticazione*, l'utente invierà l'*ID utente*, quindi la *richiesta di password* e la risposta dell'utente con *P_{UID}*.



Supponiamo che la rete sia sicura e concentriamoci sul lato del verificatore. Il segreto è la password dell'utente e il client crea e trasmette la prova tipicamente con una funzione F che è la **funzione di identità**. La prova è la password inviata in chiaro, il che è ovviamente pericoloso.

Quando il server riceve la password, deve verificare se è corretta o meno:

- **1° caso:** se la funzione f utilizzata per memorizzare la password è la funzione identità, allora è la soluzione, perché il server conosce tutte le password in chiaro e verificarne la correttezza è semplice. È rischioso: se qualcuno copia il database, avrà accesso a tutti i dati.
- **2° caso (quello suggerito):** in questo caso f non è un'identità ma un **hash unidirezionale** (cioè un digest della password) e il server non conosce la password in chiaro, ma solo il digest. Ciò significa che il controllo degli accessi è un po' più complesso, perché quando la prova viene ricevuta l'hash della prova è confrontato con l'hash memorizzato nel database delle password. Se il database viene rubato, l'attaccante non avrà una copia delle password in chiaro.

L'autenticazione basata su password è solitamente comoda per l'utente, ma solo se deve ricordare una sola password, quindi riutilizzabile. La situazione attuale è spiacevole, perché in alcune applicazioni c'è bisogno di diverse password, che non possono essere tenute a mente da una persona, quindi dovrebbero essere memorizzate dall'utente, e questo è fonte di insicurezza. Gli **svantaggi** dell'autenticazione basata su password sono:

- La **memorizzazione della password lato utente**: può essere scritta su un post-it o su un gestore di password lato client (chiamato anche password wallet), che la memorizza in modo criptato utilizzando di solito una sola passphrase;
- **Password indovinabili**;
- **Memorizzazione della password sul lato server**: il server deve conoscere la password *in chiaro* o un suo digest non protetto ([attacco a dizionario](#));
- La password può essere **sniffata** mentre viene inviata attraverso la rete;
- Potrebbero verificarsi **attacchi** al DB delle password presso il verificatore (se il DB contiene testo in chiaro o solo password offuscate);
- **Indovinare la password**: è molto pericoloso se può essere fatto offline, ad esempio con un elenco di hash di password;
- **Duplicazione della password**: utilizzo della stessa password per diversi servizi, a causa del riutilizzo della password dell'utente. Questo potrebbe essere un problema perché se l'utente ha la stessa password per un servizio ad alta sicurezza e per uno più debole, un attaccante potrebbe scoprirla sul sistema più debole e avere così accesso a quello ad alta sicurezza;
- **Invecchiamento della crittografia**: la soluzione adottata per la verifica del segreto non deve essere legata a uno specifico algoritmo crittografico, perché potrebbe essere difficile adattarsi alla necessità di cambiare l'algoritmo utilizzato, a causa di nuovi attacchi e maggiore potenza di calcolo;
- **Acquisizione di password** tramite [spoofing](#) del server e [phishing](#);
- **Attacchi MITM**.

La migliore prassi per le password è:

- È necessario utilizzare un **mix** di caratteri alfabetici (maiuscoli e minuscoli), comprese le cifre e i caratteri speciali. Purtroppo, molti sistemi non consentono l'uso di caratteri speciali o pongono un limite alla lunghezza della password;
- Utilizzate una password **lunga**, di almeno 8 caratteri;
- **Non utilizzate mai le parole del dizionario**, perché gli aggressori utilizzano i dizionari di tutte le lingue;
- **Cambiare frequentemente la password**: se la stessa password viene mantenuta per un anno, l'attaccante ha più tempo per eseguire i suoi calcoli. È sufficiente cambiare la password una o due volte all'anno per ridurre la finestra di esposizione;
- **Si cerca di non utilizzare le password**, ma è inevitabile a meno che non si utilizzino

tecniche biometriche. Memorizzazione della password sul **lato server**:

- **Non** memorizzatela **mai** in chiaro;
- Se la **password è criptata**, il server deve conoscere la chiave in chiaro, il che può essere un

problema. La soluzione è quella di **memorizzare un digest** della password, ma attenzione all'[attacco a dizionario](#) che può essere reso più veloce da una tecnica chiamata [rainbow table](#). Per evitare questo tipo di attacchi dobbiamo inserire una variazione inaspettata, chiamata *sale*.

Memorizzazione della password sul **lato client**:

- Dovrebbe essere solo nella testa dell'utente;
- Se le password sono molte, utilizzate un file criptato o un portafoglio di password.

L'"attacco del dizionario"

Se si memorizza l'hash semplice della password, sono possibili *attacchi a dizionario*. Ciò è possibile in due ipotesi:

1. **algoritmo di hash noto** utilizzato dal server;
2. perdita di informazioni, in modo che l'aggressore abbia una copia dei **valori hash** delle **password**.

Gli hash non sono funzioni invertibili, ma è possibile effettuare un **pre-computo**, quindi se non esiste ancora una copia di un hash di una password è possibile decidere che in futuro sarebbe bello attaccare le password memorizzate come hash SHA1 semplice.

Si deve ottenere un dizionario contenente non solo le lingue italiane, ma tutte le lingue possibili. Per ogni parola del dizionario, si calcola l'hash della parola e lo si memorizza in un DB abbinato alla parola corrispondente; con "parola" si intende una possibile passphrase, non una parte di essa. In genere, gli aggressori hanno un dizionario esteso a parole come nomi di personaggi famosi.

L'ipotesi principale è che l'utente insegua una delle parole contenute nel dizionario. L'attacco è il seguente:

1. A un certo punto l'attaccante ottiene un valore hash, a causa di una perdita;
2. L'attaccante esegue una semplice **ricerca** come segue: $w = \text{lookup}(\text{DB}, \text{HP})$, dove DB è il database e HP è l'hash calcolato di una parola nel database, se una qualsiasi password dell'hash compare in una tupla;
3. Se la risposta è positiva, la password corrisponde a quella parola. In caso contrario, la password non corrisponde al dizionario utilizzato.

La chiave è il **precalcolo**, perché se si aspetta di ottenere una copia dell'hash della password e solo a quel punto si inizia a calcolare tutti i possibili valori dell'hash, potrebbe essere troppo tardi, perché la password potrebbe essere cambiata.

Tavolo arcobaleno

L'attacco a dizionario può essere reso più veloce ed efficace con la tecnica della **tabella Rainbow**. Si tratta sempre di un *attacco a dizionario*, ma si tratta di un compromesso tra spazio e tempo. Provando tutte le password possibili, il calcolo dell'hash sarebbe veloce, ma il risultato sarebbe un database enorme. Se si dispone di un database completo, la ricerca sarebbe veloce, ma in questo caso vengono memorizzate meno password e viene impiegato un po' più di tempo per calcolare la password se è presente l'hash corrispondente. Si tratta di un miglioramento perché rende fattibile l'attacco esaustivo per alcuni insiemi di password.

Immaginate di creare una tabella arcobaleno per attaccare una password che sappiamo contenere 12 cifre. L'attacco esaustivo richiederebbe 10^{12} righe, ovvero un numero enorme di righe (contenenti le password e i corrispondenti valori hash). La tabella Rainbow può essere utilizzata per ridurre il numero di righe nel database di un fattore 1000. In questo modo si ottiene un database di 10^9 righe, dove ogni riga rappresenta 1000 password. Per ottenere questo risultato si utilizza la **funzione di riduzione**:

$$r: h \rightarrow p$$

È una funzione r che prende in input un hash e crea una possibile password. **Attenzione: NON è l'inverso dell'hash**, perché l'inverso dell'hash non esiste. È solo una *funzione di mappatura* che da un hash crea una delle possibili password dell'intero insieme.

Il **precomputo** è quindi il seguente:

1. Selezionare 10^9 password distinte (la dimensione desiderata) chiamate P ;
2. Per ognuna di esse inizializziamo il calcolo partendo da quella specifica password, e poi iteriamo per 1000 volte: ogni volta viene calcolato l'hash della password corrente (chiamato k) e poi si utilizza la *funzione di riduzione* per passare da k a un'altra possibile password;
3. Alla fine, la password P del primo ciclo viene memorizzata nel database insieme all'ultimo calcolo della funzione di riduzione (chiamata p).
4. Quindi, la voce rappresenta implicitamente tutte le 1000 password provate. Si noti che non ci sono più hash da memorizzare.

Poi l'**attacco** si alza in questo modo:

1. HP è l'hash trapelato di una password;
2. Avvia un'iterazione di 1000 volte al massimo, e ogni volta viene utilizzata la *funzione di riduzione* per passare dal valore dell'hash a una possibile password;
3. Quindi, cercare nel database se esiste una riga in cui p (il calcolo della funzione) è la fine della catena. In questo caso, abbiamo trovato la catena contenente quell'hash, altrimenti viene calcolato un nuovo valore k eseguendo l'hash della password;
4. Dopo aver trovato la catena, occorre calcolare nuovamente gli hash per vedere qual è l'hash che corrisponde a quello in nostro possesso.

Il problema è che, poiché la *funzione di riduzione* passa da un hash a una possibile password, potrebbero esserci due hash diversi che generano la stessa password, e questo si chiama **fusione**. Invece di usare una funzione di riduzione, si usa un insieme di n funzioni di riduzione, una per ogni fase di riduzione. Su Internet sono in vendita tabelle arcobaleno precalcolate per varie funzioni hash e set di password (ad esempio, SHA1 per le password alfanumeriche).

Questa tecnica è utilizzata da diversi programmi di attacco.

Utilizzo del sale nella memorizzazione della password

Il punto critico di questo tipo di attacco è che l'attaccante esegue un pre-computo. Senza la tabella arcobaleno e senza il database creato dal dizionario, l'operazione richiederebbe molto tempo. Per questo motivo, dobbiamo evitare il pre-computo dell'attaccante, e questo può essere fatto con un semplice trucco: **non dare all'attaccante le informazioni per il pre-computo**, perché si basa sull'idea che l'attaccante possa sapere qual è la password (attraverso il dizionario).

Utilizzando la seguente tecnica, anche se è possibile indovinare una possibile password, l'attaccante non ottiene la tabella hash perché ogni volta che viene creato un ID utente, il sistema genera un **sale** che è diverso per ogni utente. Il sale è una stringa di byte casuale (imprevedibile) e lunga (maggiore complessità del dizionario).

Gli utenti non devono memorizzare il sale, che dovrebbe contenere caratteri di uso raro o di controllo. L'hash viene quindi calcolato utilizzando la password concatenata con il sale: $HP = \text{hash}(\text{pwd} \mid \text{salt})$. Il verificatore memorizza l'**UID**, **HPUID** e **saltUID**. Se qualcuno ottiene le informazioni nel database, ottiene anche il sale, ma solo a quel punto

■ **pre-computation:**

- for (10^9 distinct P)
- for ($p=P$, $n=0$; $n < 1000$; $n++$)
 - $k = h(p)$; $p = r(k)$;
- store (DB, P , p) // chain head and tail

■ **attack:**

- let HP be the hash of a password
- for ($k=HP$; $n=0$; $n < 1000$; $n++$)
 - $p = r(k)$
 - if lookup(DB, x , p) then exit ("chain found!")
 - $k = h(p)$
- exit ("HP is not in any chain of mine")

può iniziare il calcolo, che richiederà molto tempo e nel frattempo la password potrebbe essere stata modificata. Inoltre, esistono diversi HP per gli utenti che hanno la stessa password.

Questo rende quasi impossibili gli attacchi a dizionario (compresi quelli basati sulle tabelle arcobaleno).



L'attacco di LinkedIn

Nel giugno 2012 qualcuno è riuscito a copiare 6,5 milioni di password da LinkedIn, che erano **hash SHA-1 semplici e non salati**. Questa persona ha pubblicato questi hash su Internet e ha chiesto il *crowdsourcing* per la decifrazione cooperativa delle password (il che significa: provare a calcolare l'hash SHA-1 delle parole e vedere se qualcuno ha una corrispondenza). Sono state trovate almeno 236.578 password prima che l'Interpol riuscisse a bandire il sito web che aveva pubblicato gli hash delle password.

Contemporaneamente LinkedIn ha scoperto che l'app LinkedIn per iPad/iPhone inviava dati sensibili in chiaro (non rilevanti per LinkedIn!).

Case History: password in MySQL

MySQL è un database in cui nome utente e password sono memorizzati nella tabella "user". MySQL (dalla versione 4.1) utilizza un **doppio hash (ma senza sale!)** per memorizzare la password: $SHA1(SHA1(password))$. Quindi, viene memorizzata la codifica esadecimale del risultato, preceduta da * (per distinguere questo caso da MySQL < 4.1). Ad esempio:

- Per la password "Superman!!!" il campo user.password è:
* 868E8E4F0E782EA610A67B01E63EF04817F60005

Per verificare che si tratti del doppio hash della parola, è possibile utilizzare il seguente comando su Linux:

```
$ echo -n 'Superman!!!' | openssl sha1 -binary | openssl sha1 -hex  
(stdin)= 868e8e4f0e782ea610a67b01e63ef04817f60005
```

Questo è il modo standard con cui MySQL memorizza le password, il che non va bene. Una buona cosa è cambiare il modo standard con cui MySQL memorizza le password, utilizzandone una salata.

Case History: ransomware per iPhone

Nel maggio 2014 l'account iCloud (con autenticazione a 1 fattore) è stato violato. Qualcuno è riuscito a scoprire le password e una volta entrato nell'account ha utilizzato la funzione "*Blocco remoto*" per **bloccare gli smartphone**. Poi è stato inviato un messaggio alla vittima che diceva: "*Dispositivo violato da Oleg Pliss! Per riprendere il controllo, inviare 100 USD/EUR tramite PayPal a lock404@hotmail.com*".

L'unico modo possibile era utilizzare la *modalità di recupero* che cancella tutti i dati e le applicazioni. Si è trattato di un attacco distruttivo, poiché l'aggressore non voleva guadagnare denaro (i pagamenti PayPal sarebbero stati comunque tracciati), ma ha chiesto alle persone di pagare anche se l'account era falso.

Autenticazione forte (tra pari)

Recentemente è emerso l'obbligo di non usare l'autenticazione standard, ma di usare l'autenticazione forte tra pari. Viene sempre richiesta nelle specifiche, ma non viene mai definita formalmente, oppure viene definita in troppi modi (il che è inutile).

Definizione della BCE per l'Internet Banking

Secondo la BCE (Banca Centrale Europea), un'autenticazione forte del cliente è una procedura basata sull'uso di due o più elementi: conoscenza, proprietà ed ereditarietà. Gli elementi selezionati devono essere reciprocamente indipendenti, in modo che la violazione di uno non comprometta gli altri. Almeno un elemento deve essere non riutilizzabile e non replicabile (ad eccezione dell'ereditarietà), e non deve poter essere rubato surrettiziamente via Internet. La procedura di autenticazione forte deve essere progettata in modo da proteggere la riservatezza dei dati di autenticazione (se si utilizza una password, non è possibile inviarla in chiaro).

Definizione PCI-DSS per i pagamenti con carte di credito

Secondo la definizione PCI-DSS, che riguarda i pagamenti con carte di credito, a partire dalla versione 3.2 è richiesta l'**autenticazione a più fattori (MFA)** per l'accesso all'ambiente dei dati dei titolari di carta (CDE): non importa se si tratta di una rete fidata o non fidata, e viene utilizzata anche quando l'accesso viene

effettuato dagli amministratori. L'unica eccezione è rappresentata dall'*accesso diretto alla console* (sicurezza fisica), il che significa che si entra nella stanza in cui si trova il computer.

server è posizionato. Per l'accesso remoto è sempre richiesto da reti non fidate e da utenti e terze parti (come la manutenzione).

Questa era la prassi migliore fino al gennaio 18 e in seguito è stata resa obbligatoria.

Ricordate: L'MFA non consiste nell'utilizzare due volte lo stesso fattore

(ad esempio, due password). Altre definizioni

Secondo l'*Handbook of Applied Cryptography*, l'autenticazione è una **sfida-risposta crittografica**.

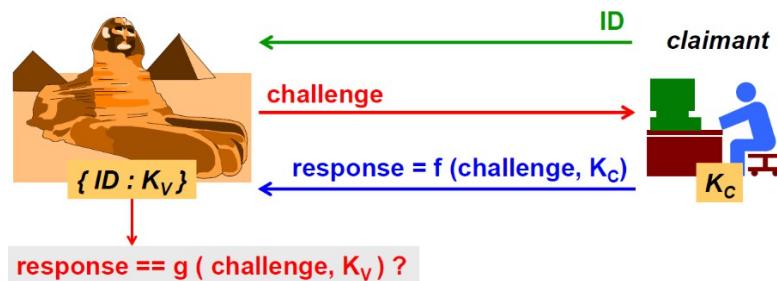
protocollo di identificazione. Più in generale, si tratta di una tecnica che resiste a un insieme ben definito di attacchi. Una tecnica authN può essere considerata forte o debole a seconda del modello di attacco:

- Ad esempio, gli utenti dell'Internet Banking → definizione della BCE;
- Ad esempio, dipendenti del PSP → definizione PCI-DSS.

In generale, è necessario prestare attenzione al proprio campo di applicazione specifico, perché questo indica il tipo di rischi e il tipo di forza che l'autenticazione forte deve avere.

Autenticazione a risposta multipla (CRA)

Il protocollo Challenge-response è un modo possibile per implementare l'autenticazione forte. CRA significa che il verificatore invia una sfida al *richiedente*. Il richiedente risponde con la soluzione calcolata utilizzando una conoscenza segreta e la sfida. Il verificatore confronta la risposta con una soluzione calcolata tramite un segreto associato al richiedente.

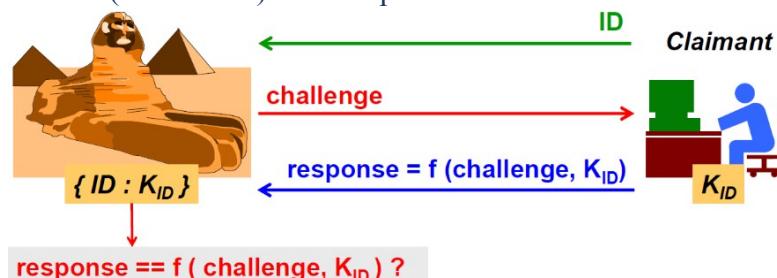


Qualcuno dichiara di possedere un identificatore (*ID*). Il verificatore controlla che esista una riga associata a quell'*ID* e invia al richiedente la *sfida*. Il richiedente possiede una chiave (*Kc*) e la utilizza per eseguire alcune tipo di calcolo (funzione *f*) e genera una risposta. La risposta può essere verificata applicando la funzione *g* a

la sfida e a una chiave nota (*KV*) del verificatore. Le chiavi possono essere diverse o uguali.

- La sfida deve essere non ripetibile per evitare attacchi replay. Per questo motivo, di solito la sfida è un *nonce* (casuale).
- La funzione *f* deve essere non invertibile, altrimenti un ascoltatore può registrare il traffico e trovare facilmente il segreto condiviso utilizzando la funzione $K_c = f^{-1}(\text{response}, \text{challenge})$

Sistemi (simmetrici) sfida-risposta



In questo caso, c'è una chiave comune condivisa tra richiedente e verificatore, che di solito è la password o la passphrase dell'utente. La funzione *f* viene calcolata due volte: una volta dall'utente per effettuare la risposta e una volta dal verificatore per verificare la corrispondenza.

I problemi generali del **CRA simmetrico** sono i seguenti:

- L'implementazione più semplice utilizza una funzione di hash (più veloce della crittografia) come SHA1 (deprecato), SHA2 (consigliato) o SHA3 (futuro);
- *KID* deve essere noto in chiaro al verificatore e questo può portare ad attacchi contro la tabella $\{ID : K_ID\}$ del verificatore;

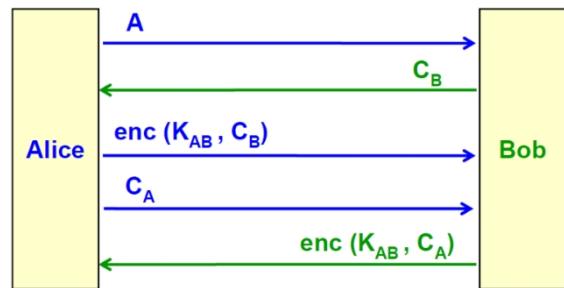
Esiste una tecnica chiamata **SCRAM** (Salted CRA Mechanism) che risolve questo problema utilizzando password con hash presso il verificatore, che offre anche il channel binding e la mutua autenticazione, mentre noi parliamo sempre di autenticazione singola.

Autenticazione reciproca con sfida simmetrica (v1)

Immaginiamo di non usare l'hash ma la crittografia (con l'hash funzionerebbe allo stesso modo) e immaginiamo di volerci autenticare reciprocamente.

Alice e Bob hanno una chiave condivisa K_{AB} e Alice invia a Bob un messaggio A (che significa: "Ehi, sono Alice!"). Bob risponde con una sfida C_B e Alice risponde con la chiave condivisa K_{AB} .

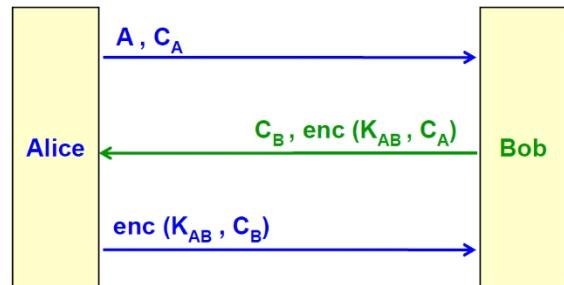
$\text{enc}(K_{AB}, CB)$ che è la crittografia della sfida usando la chiave condivisa. Allora anche Alice potrebbe lanciare una sfida (C_A). e Bob risponderebbe allo stesso modo facendo $\text{enc}(K_{AB}, CA)$. In questo modo si ottiene una protezione contro gli attacchi MITM, perché se le sfide sono nonces il Replay Attack non è possibile.



Autenticazione reciproca con sfida simmetrica (v2)

IBM, nel suo sistema di rete SNA, ha utilizzato la stessa tecnica con un'implementazione diversa: ha ridotto il numero di messaggi per ottenere prestazioni migliori ma senza alcun impatto sulla sicurezza.

Nella prima fase Alice invia sia l'identità (A) che la sfida (C_A). La risposta di Bob contiene sia la sfida (C_B) sia la crittografia $\text{enc}(K_{AB}, CA)$. Infine, Alice risponde con l' $\text{enc}(K_{AB}, CB)$. Questo sembra equivalente al precedente in termini di funzionalità e sicurezza, ma non è così.



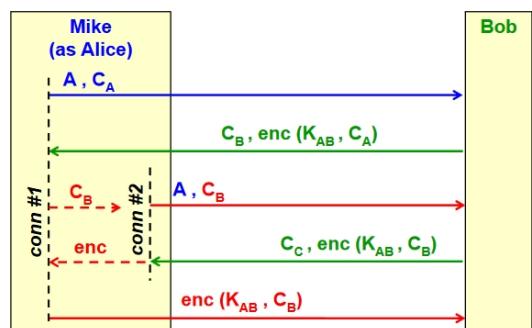
Attacco al protocollo di sfida simmetrico (v2)

Qui c'è Mike che finge di essere Alice. Mike invia a Bob l'identità di Alice (A) e la sfida di Alice (C_A). Bob risponde con C_B e $\text{enc}(K_{AB}, CA)$.

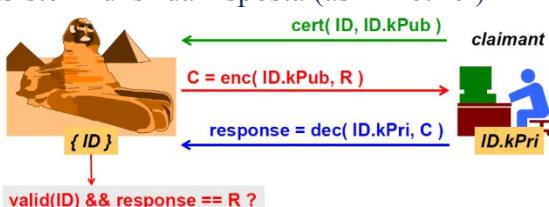
A questo punto, Mike non conosce K_{AB} e non può calcolare la risposta alla sfida. Ma a questo punto Mike apre un nuova connessione con Bob, inviando nuovamente l'identità di Alice (A) ma questa volta invia la sfida inviata da Bob (C_B). Bob

risponde nuovamente con un'altra sfida C_C e $\text{enc}(K_{AB}, CB)$ che è la risposta alla sfida della connessione 1st

Mike può finalmente fornire la risposta corretta. Naturalmente, questo può essere contrastato se c'è un limite alle connessioni.



Sistemi di sfida-risposta (asimmetrici)



È possibile utilizzare anche una sfida-risposta *asimmetrica*. Questa volta l'utente non invia la sua identità, ma il suo certificato X.509, che dichiara la sua identità e la sua chiave pubblica (il richiedente possiede la corrispondente chiave privata $ID.kPri$). Il verificatore crea una sfida prendendo un nonce casuale R criptato con la chiave pubblica. Il richiedente può decifrarlo con

utilizzando la sua chiave privata e inviarla nuovamente al verificatore.

Se la risposta è uguale al valore R inserito nella sfida, il possesso della chiave privata è provato. L'unico controllo necessario è verificare se nel database del verificatore è registrato un ID valido.

Analisi

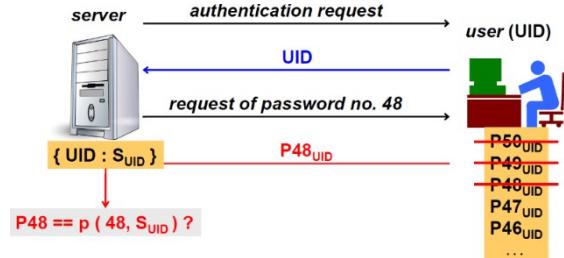
Il CRA asimmetrico è il meccanismo più forte utilizzabile, che non richiede la memorizzazione di alcun segreto presso il verificatore. Viene sempre implementato per l'*autenticazione tra pari* (client e server) in *IPsec*, *SSH* e *TLS*. È anche la chiave di volta per l'autenticazione degli utenti di un nuovo sistema di autenticazione chiamato FIDO.

Presenta alcuni problemi come:

- **La lentezza**, che deriva dal fatto che la crittografia asimmetrica è generalmente lenta;
- Se progettato in modo impreciso, può portare a una **firma involontaria da parte del richiedente**, perché quest'ultimo sta ottenendo qualcosa e sta eseguendo una computazione con la chiave privata, ma la computazione con la chiave privata ha due applicazioni: una per decifrare, l'altra per creare una firma. Quindi, c'è il rischio di firmare un documento. Per questo motivo, i dati forniti come input alla decrittazione dal lato del richiedente devono avere un formato specifico;
- **Problemi di PKI** (*radice fidata, vincolo del nome, revoca*) perché utilizza i certificati. Ciò è evitabile se il verificatore memorizza kPubID (spostando un impegno PKI equivalente al verificatore). Questo è l'approccio seguito da SSH: prendere la chiave pubblica e pubblicarla sul server, con il rischio che qualcuno possa cambiare la chiave memorizzata sul server.

Password unica (OTP)

Invece di avere password riutilizzabili, gli utenti hanno password valide solo per un accesso. L'utente che vuole accedere al server riceve una *richiesta di autenticazione*. L'utente risponde utilizzando la sua userID. A questo punto il server richiede una password specifica tra un lungo elenco (ad esempio, la n. 48). Se $P_{49,UID}$ e $P_{50,UID}$ sono già state utilizzate, l'utente invia $P_{48,UID}$ e lo pone come usato sul tavolo.



A questo punto il server dispone di una tabella contenente gli utenti (*UID*) e il segreto (*SUID*) che l'utente non conosce. Il segreto è stato utilizzato per generare le password. A questo punto, il Verificatore eseguirà la stessa funzione

utilizzato per generare la password per generarla di nuovo e verificare se corrisponde a quella inviata dall'utente. Questo rende la password **immune da sniffing**. La password può essere inviata in chiaro perché la prossima volta sarà un'altra.

Sintesi

- La password è **valida solo per un'esecuzione** del protocollo di autenticazione; l'esecuzione successiva richiede un'altra password;
- Pertanto, è **immune da sniffing**;
- È **soggetto a MITM** in quanto qualcuno si sostituisce al verificatore: per evitare questo attacco è necessaria l'autenticazione del verificatore;
- Il **provisioning** per gli abbonati è **difficile**: sono richieste molte password e il set di password sarà vuoto a un certo punto;
- L'**inserimento della password** è **difficile** anche perché di solito si tratta di una sequenza di caratteri casuali, per evitare di indovinare.

Fornitura di OTP agli utenti

Se si ipotizza che il richiedente stia lavorando su un *dispositivo stupido*, cioè privo di capacità di calcolo, o su una postazione di lavoro insicura/non attendibile, allora in questi casi si dovrebbe ricorrere a qualche sicurezza fisica aggiuntiva. Ad esempio:

- Foglio di carta contenente password precalcolate;
- Autenticatore hardware (crypto token): un oggetto in cui vengono memorizzate o generate le

password.

Al contrario, se si lavora su una postazione di lavoro intelligente e sicura possono essere **calcolati automaticamente**.

da un'applicazione ad hoc (tipica di smartphone, tablet, laptop, ...).

Il sistema S/KEY

Questa è stata la prima definizione e implementazione di OTP da parte dei Bell Labs (1981).

- L'utente genera un segreto SID ;
- Quindi l'utente calcola autonomamente N one-time password dove $P_1 = h(SID)$, $P_2 = h(P_1)$, ..., $P_N = h(P_{N-1})$ (hash associato)
- Il Verificatore memorizza l'ultima P_N . Questa password non verrà mai utilizzata direttamente per l'autenticazione, ma solo indirettamente.
- Quando l'utente vuole accedere al server, il Verificatore (che ha ottenuto P_N) chiede P_{N-1} e ottiene X dall'utente. Quindi viene eseguito il seguente controllo:

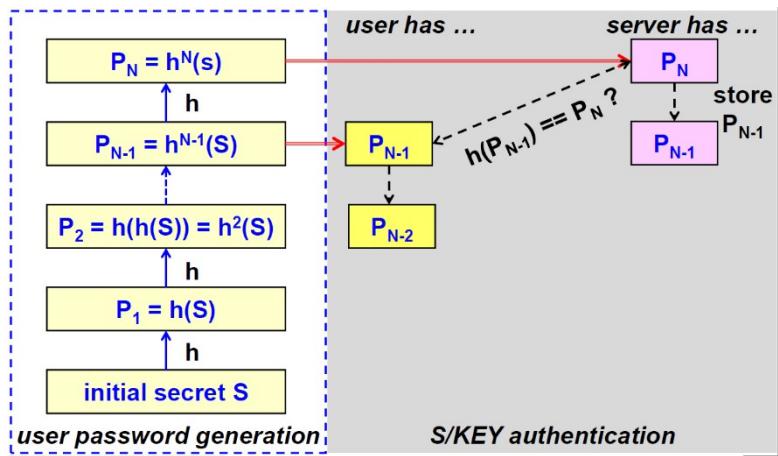
$$\text{if } (P_N \neq h(X)) \text{ then}$$

$$FAIL$$

$$\text{else}$$

$$OK; \text{ store } X \text{ as } P_{N-1}$$

Il trucco è che il Verificatore chiede le password in ordine **inverso**. In questo modo il Verificatore non ha bisogno di conoscere il segreto dell'utente e solo l'utente deve conoscere tutte le password. È documentato nella *RFC-1760* e utilizza MD4 (sono possibili altre scelte) e S/KEY è un esempio di OTP off-line / precalcolato.



Nota: il MITM è sempre possibile per l'OTP, pertanto S/KEY deve essere utilizzato insieme all'autenticazione del server.

Figura 39 Generazione e verifica della password con il sistema S/KEY

S/KEY - Generazione dell'elenco di password

L'utente inserisce una frase d'accesso (PP) che deve essere lunga almeno 8 caratteri e deve essere segreta, altrimenti se viene rivelata la sicurezza di S/KEY è compromessa. La PP viene concatenata con un seme fornito dal server che non è segreto (invia in chiaro da S a C). Questo permette di utilizzare lo stesso PP per più server (utilizzando semi diversi) perché è la combinazione tra PP e seme, ed è anche possibile riutilizzare in modo sicuro lo stesso PP cambiando il seme.

Dall'hash MD4 viene estratta una quantità di 64 bit che genera 128 bit (facendo lo XOR tra il primo/terzo gruppo di 32 bit e il secondo/quarto gruppo).

S/KEY - password

Le password a 64 bit sono un compromesso, ma dovete ancora inserirle. Inserire 64 bit come caratteri esadecimali significa 16 caratteri esadecimali. Quindi, normalmente vengono inserite come una sequenza di sei brevi parole inglese scelte da un dizionario di 2048 (e.g. 0 = A, 1 = ABE, 2 = ACE, 3 = ACT, 4 = AD, 5 = ADA). Si tratta di selezionare 11 bit dall'hash calcolato e di utilizzare un dizionario con alcune parole semplici corrispondenti alle combinazioni (2048 in questo caso); client e server **devono condividere lo stesso dizionario**. Ad esempio:

- **Password (testo):** YOU SING A NICE OLD SONG, ma non perché sia una password, bensì perché "YOU" è una delle parole del dizionario e rappresenta 11 bit. In totale ci sono 6 parole, il che significa 66 bit (più di 64, quindi va bene).

- **Password (numerica):** $1D6E5001884BD711$ (esadecimale) o $2, 120, 720, 442, 049, 943, 313$ (*decimal*)

Questo è solo un esempio di come codificare in modo semplice una lunga stringa di bit.

Problemi OTP

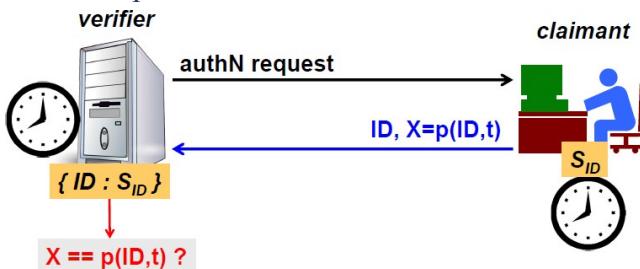
- L'OTP è generalmente **scomodo**: non è facile inserire quelle strane password;
- **Scomodo** se utilizzato per accedere a più servizi basati su password (ad esempio, POP con controllo periodico della casella di posta);
- **Costoso** se basato su autenticatori hardware: oggi è anche possibile utilizzare un software installato su uno smartphone, ma questo solleva la questione se lo smartphone sia sicuro o meno;
- Le password **cartacee** non possono essere utilizzate da un processo ma solo da un operatore umano;
- Generazione di buone password casuali e attenzione alla funzione di hash (MD4 non va più bene);
- **Fornitura di password** (generatore? SMS?);
- Quando si usa l'**ora** (come base) per generare l'OTP, deve esserci una **sincronizzazione dell'ora** tra l'utente e il sistema: se non hanno la stessa ora, si potrebbe verificare un DoS (non si riesce ad accedere perché si invia sempre una password sbagliata) o un MITM, se l'aggressore fa credere all'utente che sono le 17:00, quindi l'utente usa quell'ora per creare la password e poi l'aggressore la usa alle 17:00.

Problemi degli autenticatori hardware

Il dispositivo che crea le password potrebbe avere qualche problema:

- **Denial-of-service**: tentativi deliberatamente errati di attivare il blocco dell'account;
- **Social engineering**: se l'autenticatore non è un dispositivo fisico dedicato, ma è invece un software sullo smartphone dell'utente, potrebbe essere possibile effettuare una telefonata per simulare la perdita dell'autenticatore e inizializzarne uno nuovo da remoto.

OTP a tempo



In questa soluzione la password dipende dal tempo e dal segreto dell'utente: $p(ID, t) = h(t, SID)$. Quando il richiedente che vuole autenticarsi riceve una richiesta dal verificatore e presenterà il proprio ID più il valore generato da un dispositivo (autenticatore) che indica al richiedente quale sia la password corretta da inserire ora. A quel punto, il verificatore deve controllare se il valore è l'OTP corrente per il momento corrente. Poiché il verificatore dispone di una grande tabella contenente per ogni utente il segreto corrispondente, può eseguire lo stesso calcolo e confrontarlo.

Questo tipo di OTP richiede il calcolo locale presso l'abbonato e la sincronizzazione dell'orologio (o il mantenimento del time-shift per ogni abbonato). Dato che la password ha bisogno di tempo per essere inviata, è necessario quantizzare il tempo (cioè considerare dei timeslot, di solito lunghi da 30 a 60 secondi) e una finestra di autenticazione: di solito questo viene fatto dal verificatore considerando corretta la password generata un time slot prima e uno dopo il time slot corretto in base al proprio tempo: in formule, l'autenticazione ha successo se $X = p(ID, t) \text{ || } X == p(ID, t - 1) \text{ || } X == p(ID, t + 1)$ (dove t è il timeslot). In genere, solo uno

L'autenticazione viene eseguita per fascia oraria, il che potrebbe non andare bene per alcuni servizi, ad esempio per un broker che ha bisogno di eseguire molte transazioni alla volta.

Questo sistema è soggetto ad attacchi contro l'abbonato e il verificatore: di solito i server e i client ricevono l'ora da una fonte esterna e un attacco può essere effettuato utilizzando un falso server NTP o una femtocellula della rete mobile. Ad esempio, se l'aggressore riesce a ingannare l'abbonato e a fornirgli una password per una fascia oraria futura, può utilizzarla in quel momento.

Poiché il Verificatore memorizza il segreto di ogni utente, deve gestire un database molto sensibile: se viene rubato, l'attaccante può impersonare e calcolare la password di qualsiasi utente. Un attacco di questo tipo è avvenuto contro un sistema TOTP chiamato RSA SecurID.

Un esempio di TOTP: RSA SecurID

In questo sistema il richiedente invia al verificatore in chiaro la tripla $\{ \text{user} , \text{PIN} , \text{tokencode}(\text{seed}, \text{time}) \}$: poiché RSA SecurID è un dispositivo fisico che visualizza continuamente la password corretta, se qualcuno di diverso dall'abbonato la legge e poi la utilizza, un aggressore potrebbe entrare in possesso dell'identità dell'abbonato, il che potrebbe causare un'interruzione del servizio.

significa che è necessario un altro fattore di autenticazione. Per questo motivo, un fattore è "possedere il dispositivo", mentre l'altro è "conoscere la password (riutilizzabile)".

Poiché il PIN potrebbe essere sniffato durante l'invio attraverso la rete, esiste una variante dell'autenticatore SecurID che include un "pin pad" (una sorta di piccola tastiera solo per l'inserimento del PIN), e quando il PIN viene inserito, il valore del PIN viene preso in considerazione per la generazione dell'OTP. In questo caso, è possibile inviare solo l'utente e il codice token modificato (la tupla $\{\text{utente}, \text{token-code}^*(\text{seed}, \text{time}, \text{PIN})\}$), che è anche funzione del PIN. Sulla base dell'utente e del PIN, il Verificatore effettua un controllo su tre possibili codici token: $TC-1$, $TC0$, $TC+1$, come qualsiasi sistema TOTP.

Inoltre, a ogni dispositivo sono associati due PIN: il primo è quello utilizzato normalmente per l'autenticazione, mentre il secondo è detto **di coercizione**.

e viene utilizzato per generare un allarme quando l'abbonato è sotto attacco (ad esempio quando l'utente è costretto da un criminale ad autenticarsi).

SecurID: architettura

Oltre a fornire l'autenticatore hardware che calcola il codice del token, RSA fornisce anche un componente chiamato **ACE** (Access Control Engine):

- Il **client ACE** è installato presso la Relying Party, ovvero il server che desidera utilizzare il sistema di autenticazione;
- Il **server ACE** implementa il verificatore.

Nell'immagine, il server ACE in alto è il verificatore e quindi dispone delle informazioni globali per convalidare un codice token. I server per il servizio sono le Relaying Party, ovvero le parti che hanno il controllo degli accessi da implementare con RSA SecurID: devono avere un'implementazione del client ACE, che serve per comunicare con il verificatore, per convalidare la credenziale inviata dall'utente. Nell'esempio a sinistra, c'è un accesso remoto a un server via SSH: l'utente che ha un normale dispositivo SecurID fornisce la tripla $\{\text{utente}, \text{PIN}, \text{TC}\}$, poi quando questo

Le informazioni ricevute dalla parte rilocante vengono inoltrate al verificatore, che a sua volta fornirà una risposta (credenziali valide o meno). A destra, è riportato un esempio di autenticazione fallita utilizzando

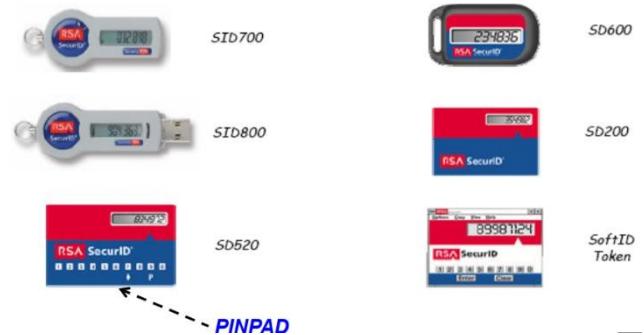


Figura 40 Diversi autenticatori forniti da RSA: il SID700 è il modello base; il SID800 è dotato di un'interfaccia USB per consentire ai programmi di leggere automaticamente il codice; l'SD520 è un dispositivo delle dimensioni di una carta di credito dotato di un pin pad; il Soft ID Token è un'applicazione software che esegue lo stesso calcolo dei dispositivi hardware, da utilizzare nel caso in cui il dispositivo utilizzato sia affidabile.

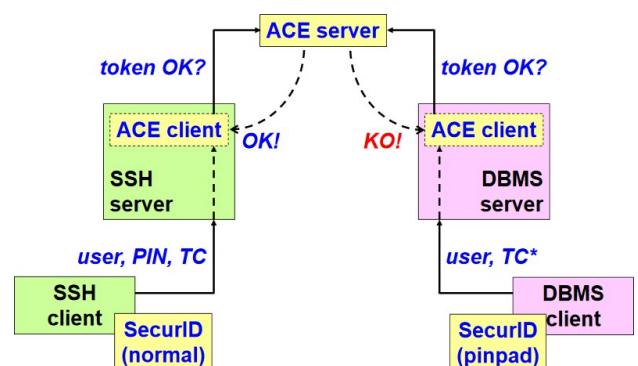


Figura 41 Struttura dell'architettura SecurID

il dispositivo con pin pad.

È importante guardare a questa architettura non solo come all'implementazione di SecurID, ma anche come a un'architettura generale da cui si evince che se è necessario un modo particolare di autenticazione, il più delle volte è necessario dotare i server di un software in grado di gestire quel nuovo tipo di autenticazione (nell'esempio il software DBMS). Se un'azienda vuole condividere la stessa tecnologia di autenticazione tra

più server, molto probabilmente è necessario un verificatore centralizzato, che fornisca la verifica a tutte le parti che si relazionano: in questo senso è possibile confrontare questo schema con quello del [NIST](#).

OTP basato su eventi

Il problema principale di TOTP è che è consentita una sola autenticazione per timeslot. Questa soluzione utilizza come input, oltre al seme, un contatore intero monotono C : $p(ID, C) = h(C, SID)$. Richiede inoltre un calcolo locale presso l'abbonato e il contatore viene incrementato presso l'abbonato (ad esempio, tramite un pulsante): in questo modo

In questo modo è possibile eseguire frequenti autenticazioni. Questo sistema permette il pre-computo della OTP (anche da parte di un avversario che ha temporaneamente accesso all'autenticatore): allo stesso tempo, il Verificatore deve adattarsi alla desincronizzazione, ad esempio perché il sottoscritto ha premuto il pulsante senza volerlo. Per fare ciò, si considera una finestra di contatori, in modo che una password sia considerata corretta se corrisponde a quella calcolata con uno di un insieme di contatori, con una dimensione fissa e solitamente piccola dell'insieme. Nelle formule, una password è considerata corretta se

$X == p(ID, C) \text{ || } X == p(ID, C + 1) \text{ || } X == p(ID, C + 2) \text{ || } \dots$ con al massimo dieci contatori successivi, che devono essere

resistente agli attacchi esaustivi. I verificatori possono gestire la desincronizzazione e memorizzare ogni volta il contatore che corrisponde alla password inviata dall'utente. Se per qualsiasi motivo nessuna delle password generate dall'utente va bene, sarà necessaria una chiamata per azzerare il contatore (ad esempio, immaginiamo che un bambino prenda il dispositivo e prema il pulsante più di dieci volte).

OTP fuori banda

Le soluzioni illustrate finora richiedono che l'utente riceva qualcosa: un elenco di password nel caso di S/KEY o un dispositivo nel caso di TOTP e EOTP: nel caso in cui queste opzioni non siano valide, ad esempio perché non è possibile fornire in modo sicuro l'elenco o non c'è fiducia nel dispositivo dell'utente, è comunque possibile utilizzare l'OTP fuori banda. Si basa sul fatto che l'OTP viene generato non utilizzando il normale canale di comunicazione. Nello schema riportato in figura, all'utente viene fornita una chiave segreta,

che è una *password riutilizzabile*. Durante l'autenticazione, l'utente invia l'id utente e la password riutilizzabile.

password: è necessaria per identificare chiaramente chi è l'utente. Le password riutilizzabili, come si è visto, non sono molto forti, ma grazie al terzo passo il verificatore può essere sicuro che l'utente si stia davvero autenticando: cerca nel suo database e recupera il numero di telefono registrato; quindi genera un OTP e lo invia (fuori banda) al cellulare dell'utente, ad esempio via SMS (quarto passo). Questo, come si può notare, avviene fuori banda perché la trasmissione dell'OTP utilizza un mezzo diverso dalla comunicazione tra il server e l'utente. Infine, nella fase cinque, l'utente può fornire l'OTP appena ricevuto.

Si tratta di un sistema ampiamente utilizzato da molte banche e da molti fornitori di identità come quelli coinvolti in SPID. Questo sistema riduce anche l'onere per l'utente, che non deve disporre di un moderno smartphone, ma solo di un dispositivo minimo per ricevere il messaggio: l'OTP viene generato dal server solo quando necessario e poi inviato in modo fidato all'utente. Attenzione però: al quinto passo è necessario un canale sicuro con autenticazione del server per evitare attacchi MITM. Il canale OOB è spesso costituito da messaggi SMS ma, poiché la maggior parte delle reti mobili è oggi implementata con VoIP, identificazione dell'utente mobile e protocollo SS7 (che sono piuttosto insicuri), il [NIST](#) suggerisce di utilizzare il meccanismo Push su canale TLS verso il dispositivo dell'abbonato registrato, inserendo il messaggio all'interno di una notifica, la cui conferma viene effettuata inserendo l'impronta digitale

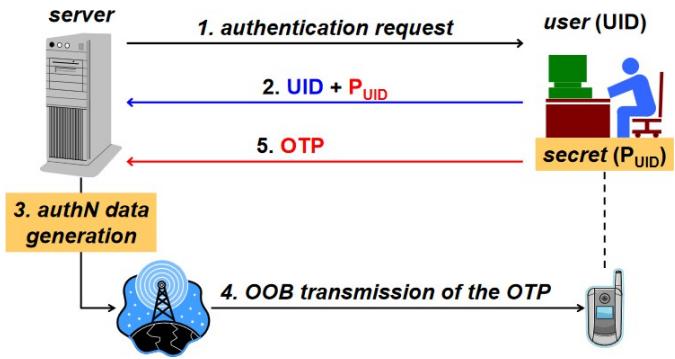


Figura 42 Le fasi di uno schema di autenticazione fuori banda

dell'utente.

AuthN a due/multi fattori (2FA/MFA)

È già stato sottolineato che, se si vuole fornire un'autenticazione forte, è necessario utilizzare più di un fattore: questo è anche chiamato **2FA** o più in generale **MFA**. L'MFA viene utilizzato sia per *aumentare la forza dell'autenticazione* sia per *proteggere l'autenticatore fisico*, come nel caso dell'OTP. Di solito, per proteggere l'autenticatore si utilizza un **PIN**:

- **PIN trasmesso insieme all'OTP:** come si è visto, il problema è che può essere sniffato quando viene inserito o trasmesso attraverso la rete;
- **PIN inserito per calcolare l'OTP stesso:** come in *RSA SecurID*;
- **PIN (o fattore di ereditarietà) utilizzato per sbloccare l'autenticatore,** molto rischioso se:
 - nessuna protezione da **tentativi di sblocco multipli**;
 - **meccanismo di blocco debole**, ad esempio il sensore di impronte digitali non è abbastanza forte da riconoscere solo l'impronta del proprietario del dispositivo;
 - **sblocco valido per una finestra temporale:** se qualcuno ha accesso all'autenticatore nella finestra temporale in cui è stato sbloccato, può utilizzare l'identità dell'utente.

Autenticazione degli esseri umani

Un altro problema che si potrebbe voler affrontare è quello di verificare che l'abbonato sia un essere umano e non un programma. Esistono due soluzioni:

- Tecniche **CAPTCHA** (Completely Automated Public Turing test to tell Computers and Humans Apart): ad esempio, una foto con immagini di caratteri distorti;
- tecniche biometriche: ad esempio, l'impronta digitale.

I sistemi biometrici e i loro problemi

L'idea principale è quella di misurare una caratteristica biologica dell'utente, come ad esempio: l'impronta digitale, la voce, la scansione della retina, la scansione dell'iride, il pattern delle vene sanguigne delle mani, la frequenza cardiaca e la geometria della mano. Tuttavia, qualunque sia la soluzione adottata, ci sono due parametri che non possono mai essere uguali a zero:

- **FAR (False Acceptance Rate):** il tasso con cui il sistema accetta come buono un segnale biometrico che in realtà non lo è;
- **FRR (False Rejection Rate):** il tasso in cui il sistema rifiuta un segnale valido come se fosse falso.

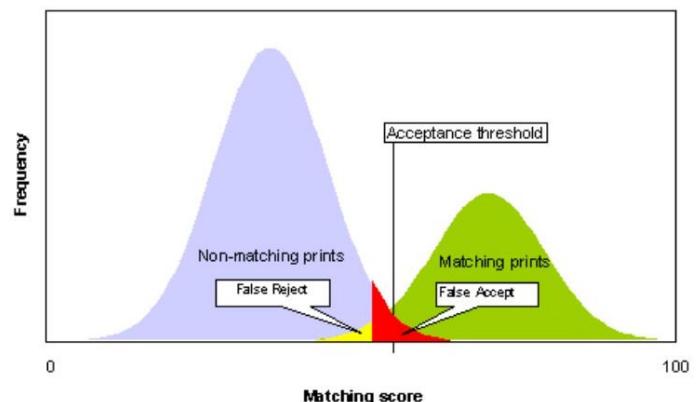


Figura 43 Problemi dei sistemi biometrici: le due distribuzioni normali si riferiscono alle impronte che corrispondono (verde) e a quelle che non corrispondono. Le due distribuzioni si sovrappongono e questo significa che è necessario porre una soglia: ovunque venga posta, FAR e FRR non possono essere entrambi uguali a zero.

Questi tassi non possono mai essere uguali a zero, perché i sistemi biometrici sono intrinsecamente imprecisi: FAR e FRR possono essere in parte regolati, ma dipendono fortemente dal costo del dispositivo. Inoltre, alcune caratteristiche biologiche sono variabili: ad esempio, l'utente può avere una ferita al dito, la voce alterata a causa di un'emozione o il pattern sanguigno della retina alterato a causa di alcol o droghe. Oltre a quelli tecnici, esistono anche altri tipi di problemi:

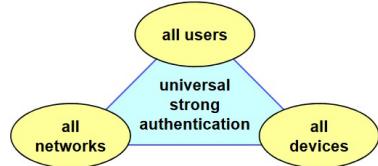
- **accettazione psicologica:**
 - **Sindrome del "Grande Fratello"**, legata alla raccolta di dati personali;
 - alcune tecnologie sono invasive e potrebbero nuocere, ad esempio le scansioni della retina possono non andare bene per alcune persone.
- **privacy**, perché a questo punto non si tratta di autenticazione, ma di identificazione e non può essere ripudiata;
- **non possono essere modificati se copiati**: una password può essere modificata se si teme che sia stata divulgata, mentre una caratteristica biometrica no. Per questo motivo, un'autenticazione biometrica non dovrebbe essere inviata attraverso la rete, ma può essere utile solo per sostituire

- localmente un PIN o una password;
- mancanza di un'API / SPI standard che porti a:
 - elevati costi di sviluppo;
 - forte dipendenza da un singolo/poco fornitore.

Interoperabilità dell'autenticazione: OATH

Come detto, le aziende che gestiscono molti server vorrebbero utilizzare la stessa autenticazione per tutte le loro relying party (cioè i server applicativi), ma ciò significa che dovrebbe essere possibile acquistare un verificatore e utilizzarlo con dispositivi creati da aziende diverse. In passato non esisteva questa interoperabilità, per cui, ad esempio, quando si acquistavano i dispositivi RSA SecurID, era necessario acquistare anche il client ACE e il server ACE di RSA. RSA non pubblicava nemmeno l'algoritmo di hash utilizzato per la generazione del token-code.

Per risolvere questo problema è nata un'iniziativa chiamata [OATH](#): l'idea è quella di fornire l'interoperabilità dei sistemi di autenticazione basati su OTP e sfide simmetriche o asimmetriche. Ciò avviene attraverso lo sviluppo di standard per il protocollo client-server e il formato dei dati sul client, al fine di ottenere un'autenticazione forte universale.



Finora, OATH ha fornito le seguenti [specifiche](#):

- **HOTP** (HMAC OTP, RFC-4226);
- **TOTP** (Time-based OTP, RFC-6238);
- **Protocollo di sfida-risposta OATH** (OCRA, RFC-6287);
- **Portable Symmetric Key Container** (PSKC, RFC-6030): quando si usa l'OTP, il segreto condiviso deve essere protetto. Questa soluzione fornisce un contenitore di chiavi basato su XML per il trasporto di chiavi simmetriche e metadati relativi alle chiavi;
- **Dynamic Symmetric Key Provisioning Protocol** (DSKPP, RFC-6063): è un protocollo client-server per la fornitura di chiavi simmetriche a un cripto-motore da parte di un server di key-provisioning.

[CALDAIA](#)

Definiamo:

- **K**: chiave segreta condivisa (tra Verificatore e Sottoscrittore);
- **C**: contatore (numero intero positivo monotono);
- **h**: funzione di hash crittografico (per impostazione predefinita è *SHA1*);
- **sel**: funzione per selezionare 4 byte da una lunga

stringa di byte. L'**OTP basato su HMAC** viene calcolato nel modo seguente:

$$HOTP(K, C) = \text{sel}(\text{HMAC} - h(K, C)) \& 0x7FFFFF$$

dove la maschera *0x7FFFFFFF* viene utilizzata per impostare MSB=0 (per evitare problemi se il risultato viene interpretato come un intero firmato). Quindi si genera un codice di accesso di N cifre (6-8):

$$\text{HOTP - code} = \text{HOTP}(K, C) \bmod 10^N$$

In questo modo implementiamo un OTP basato sugli eventi.

[TOTP](#)

Per creare un OTP basato sul tempo, la procedura è la stessa di HOTP ma il contatore C è il numero di intervalli TS trascorsi da un'origine fissa T0, in formule:

$$C = (T - T_0) / TS$$

Dove, nell'impostazione predefinita (RFC-6238), **T0** è l'epoca Unix (*1/1/1970*), **T** è *unixtime(now)* i secondi trascorsi dall'epoca Unix, **TS** è pari a *30 secondi*, **h** è SHA1 (si può usare SHA-256 o SHA-512) e **N** è uguale a 6.

Il punto importante è che con le stesse funzioni di base è possibile implementare TOTP e EOTP.

Inoltre, questo standard è importante perché Google fornisce implementazioni open-source gratuite di HOTP e TOTP, sia per il client che per il server. Nell'implementazione di Google, **K** viene fornito *codificato in base-32* (la maggior parte dei

spesso come codice QR), **C** è fornito come *uint_64*, **TS** è uguale a *30 secondi*, **N** è uguale a *6* e la funzione *sel(X)* è tale che: viene considerato un offset con i 4 bit meno significativi di X, allora restituisce *X[offset .. offset + 3]*. Se il codice generato contiene meno di 6 cifre, viene lasciato imbottito di zeri (ad esempio, 123 → 000123).

La disponibilità di questo codice da parte di Google ha spinto l'utilizzo di questo standard, oggi ampiamente diffuso.

FIDO

Uno dei tentativi più recenti di aumentare la sicurezza dell'autenticazione è lo standard denominato FIDO (Fast IDentity Online). È uno standard industriale della FIDO Alliance per:

- biometrico, che nella terminologia FIDO è chiamato *esperienza utente senza password*;
- Autenticazione a 2 fattori, che nella terminologia FIDO è chiamata *esperienza utente a 2nd fattori*.

L'obiettivo di FIDO è quindi quello di semplificare l'esperienza di autenticazione degli utenti, consentendo loro di utilizzare sistemi biometrici e più di un fattore. Il concetto principale di FIDO non è quello di inventare un altro dispositivo, ma di lasciare che l'utente sfrutti qualsiasi dispositivo personale in grado di eseguire la crittografia asimmetrica (ad esempio, laptop o smartphone), perché all'interno di FIDO verrà utilizzato:

- per rispondere a una sfida asimmetrica;
- per la firma digitale dei testi.

I protocolli FIDO sono stati progettati da zero per proteggere la privacy degli utenti. I protocolli non forniscono informazioni che possano essere utilizzate da diversi servizi online per collaborare e tracciare un utente tra i vari servizi. Le informazioni biometriche, se utilizzate, non lasciano mai il dispositivo dell'utente.

Registrazione FIDO

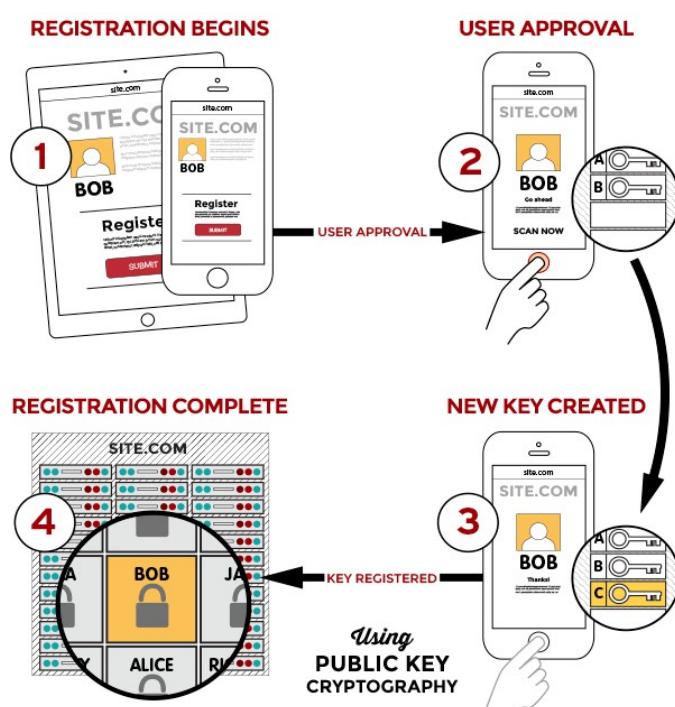


Figura 44 Procedura di registrazione FIDO, immagine tratta da <https://fidoalliance.org/how-fido-works/>

A differenza di altri sistemi, in cui la registrazione viene solitamente effettuata tramite la creazione di un nuovo nome utente e di una password classici, oppure riutilizzando l'identità dell'utente su Facebook o Google, FIDO utilizza il dispositivo dell'utente con crittografia asimmetrica. Quando ci si registra con FIDO:

1. All'utente viene richiesto di scegliere un autenticatore FIDO disponibile che corrisponda alla politica di accettazione del servizio online.
2. L'utente sblocca l'autenticatore FIDO utilizzando un lettore di impronte digitali, un pulsante su un dispositivo a secondo fattore, un PIN inserito in modo sicuro o un altro metodo.
3. Il dispositivo dell'utente crea una nuova coppia di chiavi pubbliche/private unica per il dispositivo locale, il servizio online e l'account dell'utente.
4. La chiave pubblica viene inviata al servizio online e associata all'account dell'utente. La chiave privata e tutte le informazioni sul metodo di autenticazione locale (come le misure biometriche o i modelli) non lasciano mai il dispositivo locale.

Si noti che l'identità dell'utente reale può essere fittizia! Tuttavia, la vera identità dell'utente è non è importante, lo è solo la chiave: la chiave privata è memorizzata all'interno del dispositivo e la chiave pubblica viene inviata al server web ed è associata al nome deciso per l'autenticazione. C'è l'uso della crittografia asimmetrica ma non ci sono certificati X509 (non sono necessari), perché la chiave pubblica è memorizzata sul server, associata a un nome.



Accesso FIDO

Dopo l'iscrizione, sono necessari quattro passaggi per effettuare il login:

1. Il servizio online chiede all'utente di effettuare il login con un dispositivo precedentemente registrato che corrisponde alla politica di accettazione del servizio: in questa fase l'utente fornisce il nome utente e la password (riutilizzabile);
2. L'utente sblocca l'autenticatore FIDO utilizzando lo stesso metodo utilizzato al momento della registrazione.
3. Il dispositivo utilizza l'identificativo dell'account dell'utente fornito dal servizio per selezionare la chiave corretta e firmare la sfida del servizio: questo identificativo viene recuperato considerando la coppia username-password;
4. Il dispositivo client invia la sfida firmata al servizio, che la verifica con la chiave pubblica memorizzata e registra l'utente.

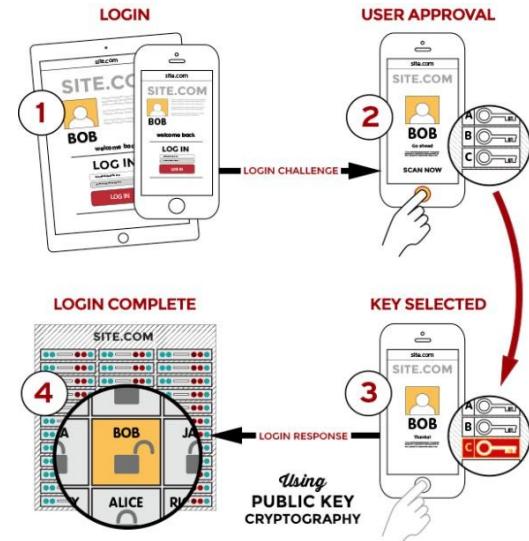
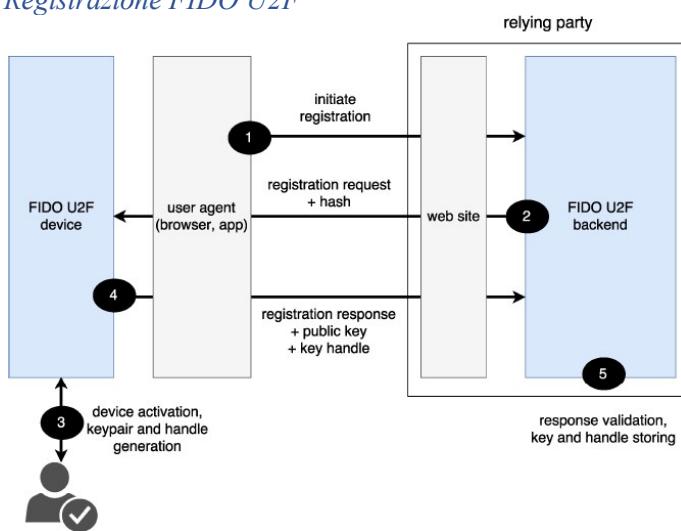


Figura 45 Procedura di login FIDO, immagine tratta da <https://fidoalliance.org/how-fido-works/>

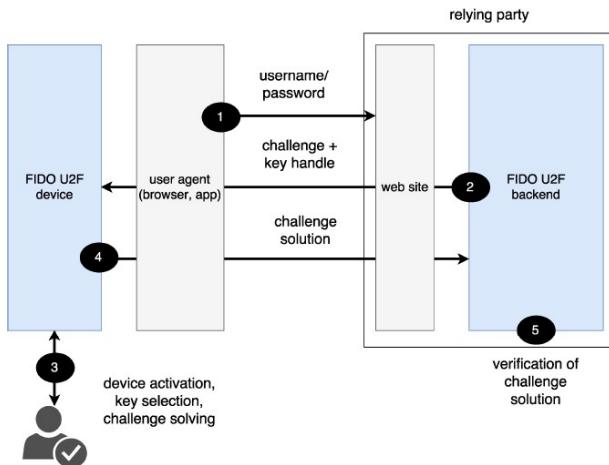
Registrazione FIDO U2F



Nella fase di registrazione, in genere il lavoro inizia con il browser o con un'applicazione che tipicamente comunica con il server. La fase di registrazione dell'utente implica che il sito web su cui gli utenti cercano di registrarsi **deve** avere un backend FIDO U2F, ovvero qualcosa in grado di gestire il protocollo FIDO. Quando arriva la richiesta di registrazione, il backend invia al dispositivo una richiesta di registrazione che include un hash (tipo nonce, per proteggere l'integrità della trasmissione). Il browser/app lo passerà al dispositivo FIDO U2F (che può essere software o hardware) e quindi l'utente dovrà **attivare esplicitamente il dispositivo**, autorizzare la **generazione della coppia di chiavi** e la **generazione dell'handle**, il che significa che l'utente deve essere in grado di gestire il protocollo FIDO U2F.

è un identificatore per questa chiave specifica (in genere l'hash della chiave pubblica). Quindi viene inviata la risposta di registrazione con la chiave pubblica e l'handle della chiave. Questi elementi vengono convalidati nella fase finale (5) e la coppia chiave-maniglia viene memorizzata.

Autenticazione FIDO U2F



L'autenticazione inizia tipicamente con il nome utente e la password (riutilizzabile) che vengono inviati al backend FIDO U2F, il quale avvia una sfida che deve essere risolta con la chiave associata al nome utente. Quando l'utente la riceve, deve attivare nuovamente il dispositivo, selezionare la chiave (in base all'handle della chiave inviato dal server) e risolvere la sfida. Quindi la sfida verrà risolta e inviata al server per la verifica.

FIDO: altre caratteristiche e analisi della sicurezza

Utilizzo di FIDO:

- **Tecniche biometriche:** metodo di autenticazione locale per abilitare le chiavi FIDO memorizzate solo sul dispositivo dell'utente;
- **Transazioni sicure:** firma digitale del testo di una transazione (oltre alla risposta alla sfida) con la stessa chiave utilizzata per l'autenticazione. Questo per evitare attacchi MITM;
- **Backend (o server) FIDO:** per abilitare l'uso di FIDO su un server applicativo.
- **Client FIDO:** per creare e gestire le credenziali FIDO su un

dispositivo utente Per quanto riguarda la sicurezza e la privacy, FIDO

fornisce:

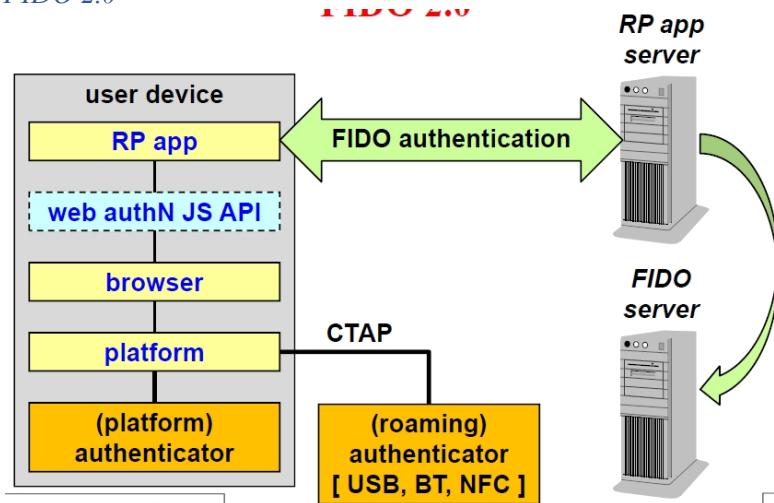
- **Autenticazione forte (*crittografia asimmetrica*):**
- **Il protocollo non prevede l'uso** di certificati X509. Si tratta di una sorta di fiducia diretta tra utente e server web;
- **Nessun segreto sul lato server:** non è esposto ad alcun attacco di riservatezza. Naturalmente, poiché memorizza le chiavi pubbliche e gli identificatori degli utenti, è necessaria l'autenticazione e l'integrità di questi dati;
- **I dati biometrici** (se utilizzati) non lasciano mai il dispositivo dell'utente;
- **Nessun phishing perché la risposta authN non può essere riutilizzata:** è una firma sul testo della transazione, compresa l'identità dell'RP (il che significa che la stessa risposta non può essere utilizzata su un altro server);
- Poiché a ogni registrazione viene generata una nuova coppia di chiavi, **non si ottiene alcuna collegabilità** tra: servizi diversi utilizzati dallo stesso utente; account diversi di proprietà dello stesso utente.
- Non c'è limite alla chiave privata generata, perché le **chiavi private non sono memorizzate nell'autenticatore**, ma vengono **ricalcolate** (generate al volo) in base a un **segreto interno** e all'**identità del RP**. Utilizzando una funzione interna appropriata con il segreto, è possibile utilizzare l'identità RP per calcolare la chiave privata.

L'evoluzione di FIDO è la seguente:

- Febbraio 2013: Lancio dell'alleanza FIDO;
- Dicembre 2014: FIDO v1.0;
- Giugno 2015: Bluetooth e NFC come trasporto per U2F;
- Novembre 2015: presentazione al W3C della Web API per l'accesso alle credenziali FIDO;
- Febbraio 2016: Il W3C crea il Web Authentication WG per definire un'API lato client che fornisca funzionalità di autenticazione forte alle applicazioni Web, basandosi sulla FIDO Web API;

- Novembre 2017: FIDO v2.0.

FIDO 2.0



Con FIDO 2.0 è stato definito un nuovo protocollo, il **CTAP (Client To Authenticator Protocol)** utilizzato per collegare un autenticatore esterno alla piattaforma. Quando inizia l'autenticazione, l'applicazione RP può utilizzare l'*API web authN JS* per accedere a FIDO. L'API viene eseguita all'interno del browser, sopra una piattaforma (come Linux, iOS, Android) e quindi ci sono due scelte: **l'autenticatore locale** (come la chiave privata memorizzata in un file su Windows) o un **dispositivo esterno** che viene abbinato al dispositivo con il **protocollo CTAP**. Questo

Il protocollo funziona tramite USB, BT, NFC, ecc. e le chiavi sono sempre presenti. Si tratta di una sorta di *autenticatore in roaming* perché il dispositivo può essere utilizzato con più dispositivi applicativi.

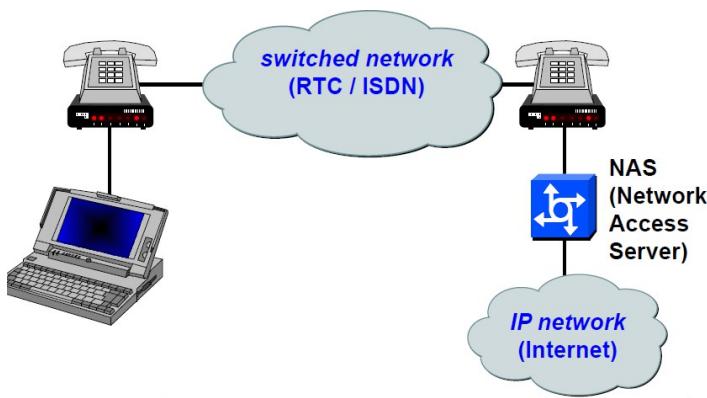
Nel caso in cui si utilizzi un autenticatore interno, esistono alcuni elementi crittografici (più o meno sicuri) in grado di memorizzare e utilizzare chiavi asimmetriche.

Ad esempio:

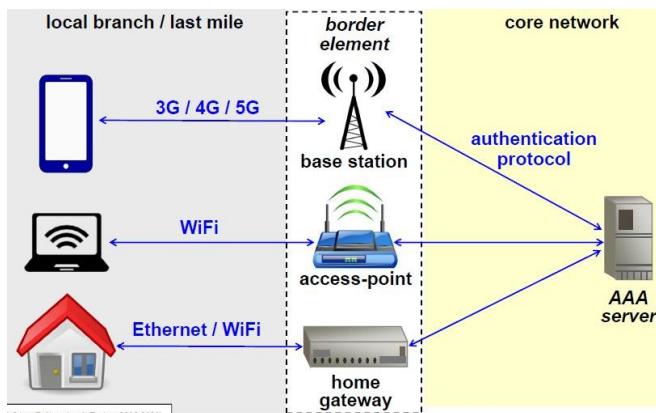
- **L'attestazione a pacchetto** è un autenticatore con alcune risorse limitate (ad esempio, un Secure Element in Android);
- **Attestazione TPM**: un chip speciale che offre alcune buone caratteristiche di sicurezza, in grado di generare e utilizzare chiavi asimmetriche;
- **Attestato chiave Android**: autenticatore disponibile a partire da Android Nougat;
- **Attestazione Android SafetyNet**: autenticatore di Android tramite API SafetyNet;
- **Attestazione FIDO U2F**: autenticatore FIDO U2F che utilizza il formato di messaggio FIDO-U2F.

Sicurezza delle reti IP

Il primo punto sulla sicurezza delle reti IP riguarda il controllo di chi può accedere alle reti. In passato, veniva utilizzata soprattutto per gli utenti residenziali, utilizzando un modem per impiegare una linea telefonica (rete commutata) per trasformare i bit in suono e avere un'apparecchiatura equivalente sul lato ISP, che accettava la chiamata telefonica e la trasformava in pacchetti di rete. Per rendere possibile tutto ciò, esistevano alcuni dispositivi chiamati **NAS** (*Network Access Server*) che avevano il compito di autenticare l'utente, effettuare il controllo degli accessi e



quindi fornire all'utente l'accesso alla rete IP (tipicamente Internet, ma in alcuni casi anche per accedere da casa alla rete interna di un'azienda). Oggi questo schema, almeno nei Paesi occidentali, non è più utilizzato, anche se alcuni Paesi lo utilizzano ancora.



Oggi è possibile accedere a Internet in diversi modi, che dipendono essenzialmente dal dispositivo. Ad esempio, uno smartphone utilizza tipicamente alcune tecnologie come 3G, 4G o 5G per connettersi alla stazione base, che esegue un **protocollo di autenticazione** con il **server AAA** per verificare se l'utente è autorizzato alla connessione a Internet. È anche possibile utilizzare il Wi-Fi per gli access-point, che fornisce la traduzione da rete wireless a rete cablata e può verificare l'accesso. Altrimenti, potrebbe esserci un gateway domestico, che può fornire l'accesso a Internet utilizzando sia l'Ethernet che il Wi-Fi (solo se

correttamente autenticato). In tutti questi schemi, è sempre necessario eseguire l'autenticazione prima di consentire il traffico di un utente specifico. Esiste quindi una differenza tra *filiale locale/ultimo miglio, elemento di confine e rete centrale*.

Autenticazione dei canali PPP

È necessario autenticare un utente prima di attivare una trasmissione di rete. Poiché l'autenticazione inizia quando qualcuno cerca di connettersi, sicuramente è già disponibile il livello 1 (fisico) per le trasmissioni fisiche. Poiché stiamo lavorando a livello logico, è disponibile anche il livello L2 (connettività del livello dati). In cima alla connettività del livello dati, normalmente viene eseguito un protocollo specifico per il trasporto di una certa quantità di dati, che di solito è il **protocollo PPP** (Point-to-point), inventato per *incapsulare i pacchetti di rete* come il livello 3 (ad esempio, IP) e *trasportarli su un collegamento PPP*. Può trattarsi di un PPP fisico, come la linea ISDN o la rete telefonica, oppure di un livello virtuale, come quando, partendo dal gateway di casa, si accede all'ADSL utilizzando **PPPoE** (*PPP over Ethernet*) per trasportare i pacchetti tra il gateway di casa e il provider. Inoltre, il PPP viene utilizzato per trasportare pacchetti all'interno di connessioni virtualizzate di livello tre con un protocollo specifico e "orribile" chiamato *L2TP* (Layer 2 Tunnel Protocol) che è un livello due incapsulato all'interno di UDP, che viola tutti i normali comportamenti di una rete. In ogni caso, una volta attivato il PPP, ci sono molti punti-punto virtuali che vanno dal dispositivo a un punto di accesso. Il PPP si attiva in tre fasi:

- **LCP** (Link Control Protocol): stabilisce la capacità di trasmettere dati;

- **Autenticazione** (opzionale; *PAP*, *CHAP* o *EAP*);
- **Incapsulamento L3** (ad esempio *IPCP*: IP Control Protocol);

Autenticazione di un accesso alla rete

Esistono tre possibilità per autenticare l'accesso alla rete tramite PPP:

- **PAP (Password Authentication Protocol)**: è il più vecchio; in questo caso l'utente invia nome utente e password in chiaro sul canale PPP. Se qualcuno sniffa il canale, acquisisce la password;
- **CHAP (Challenge Handshake Authentication Protocol)**: utilizza una risposta di sfida simmetrica basata sulla password dell'utente. In questo caso la password non può essere copiata, ma il canale non è protetto. Piuttosto che inventare altri protocolli legati a uno specifico meccanismo di autenticazione, è stata fatta una generalizzazione introducendo EAP;
- **EAP (Extensible Authentication Protocol)**: è un protocollo che **non** implementa alcun metodo. Il metodo di autenticazione è esterno: ad esempio, si possono usare challenge response, OTP o TLS.

Al giorno d'oggi PAP e CHAP non dovrebbero mai essere utilizzati, mentre EAP è il sistema più diffuso per l'autenticazione dell'accesso alla rete.

EAP

Si tratta del **PPP Extensible Authentication Protocol** (RFC-3748). È un **framework di autenticazione L2 flessibile**. È L2 perché prima di accedere a Internet (che è L3) è necessario autenticarsi. Ha meccanismi di autenticazione già predefiniti, quali: *MD5-challenge*, simmetrico e simile alla CHAP, *OTP generico* e *token card generico*. Sono stati aggiunti altri meccanismi: RFC-2716 "Protocollo di autenticazione PPP EAP TLS" e RFC-3579 "Supporto RADIUS per EAP".

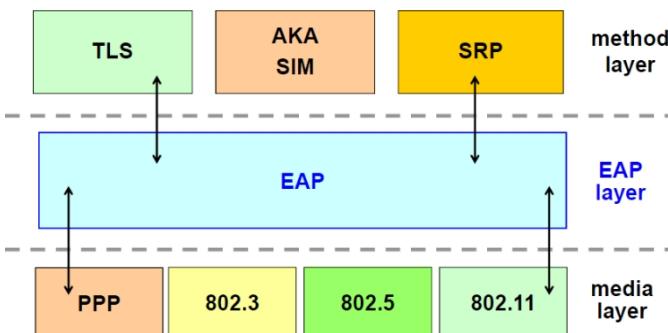
EAP ha bisogno di trasportare alcuni dati per eseguire l'autenticazione (ad esempio, trasportare un nome utente, una sfida o una risposta) ma L3 non è disponibile (perché EAP si autentica prima di L3). Per questo motivo, EAP deve creare un proprio protocollo di incapsulamento per trasportare i propri dati su L2.

- EAP definisce un piccolo protocollo L3 utilizzato solo da se stesso. Il vantaggio è che è completamente **indipendente da IP** ed è stato pensato per supportare qualsiasi livello di collegamento (EAP funziona sul vecchio PPP e sul vecchio 802.x).
- Poiché non c'è L3, fornisce l'**ACK/NAK** richiesto per i pacchetti, ma non c'è il windowing (come invece è previsto nel TCP). EAP presuppone che i pacchetti non vengano riordinati (in PPP è garantito dal protocollo), ma se si utilizza EAP su canali virtuali come UDP e IP grezzo, i datagrammi possono arrivare fuori ordine: in questo caso EAP non funzionerebbe.
- **La ritrasmissione** deve essere garantita, perché i pacchetti devono essere inviati di nuovo se vengono persi, ma deve esserci un limite nei tentativi di ritrasmissione, in genere da 3 a 5, dopo il quale l'autenticazione fallisce.
- **Nessuna frammentazione** (che dipende dalla MTU sottostante nella rete L2): deve essere curata dai metodi EAP per i payload superiori alla MTU minima EAP.

Quando l'autenticazione non funziona in EAP, non significa che l'autenticazione sia fallita, ma potrebbe esserci un problema di rete. Durante la risoluzione dei problemi EAP, potrebbe essere necessario un esperto di rete per verificare se uno di questi problemi è il colpevole del fallimento.

In EAP, non si presume che il collegamento sia fisicamente sicuro: ogni metodo di autenticazione deve garantire la sicurezza da solo. Alcuni metodi EAP sono:

- **EAP-TLS (RFC-5216)**
- **EAP-MD5 (RFC-3748)**: risposta di sfida simmetrica che fornisce solo l'autenticazione EAP tra pari, ma non l'autenticazione reciproca.
- **EAP-TTLS**: TLS a tunnel che consente di utilizzare qualsiasi metodo protetto all'interno di un canale TLS sicuro.
- **EAP-SRP (password remota sicura)**
- **GSS API** (include Kerberos)
- **AKA-SIM (RFC-4186, RFC-4187)**: Subscriber Identity Module è il sistema utilizzato nelle reti mobili.



A sinistra è riportata l'architettura generale di EAP. Nella parte inferiore è riportato qualsiasi tipo di canale L2 supportato da EAP (ad esempio, PPP), 802.3 (Ethernet), 802.5 (Token ring), 802.11 (Wi-Fi)). EAP fornisce a queste reti specifiche la disponibilità dei suoi metodi di autenticazione, come TLS, AKA-SIM, autenticazione SRP (*indipendente dalla L2*).

Autenticazione per l'accesso alla rete

L'autenticazione per l'accesso alla rete funziona come l'architettura a destra. A sinistra ci sono alcuni collegamenti di comunicazione (modem, access point o ADSL/Fibra) che a un certo punto terminano in un dispositivo ospitato dall'ISP. Questi dispositivi (per connettere tutti i possibili utenti) sono controllati da un NAS (*Network Access Server*) che riceve le richieste dai client e deve sapere se si tratta di un utente valido o meno. Utilizzerà il protocollo su

la rete IP di backend locale al NAS e poi parlerà con il server di autenticazione centralizzato. Questo perché l'ISP ha molti punti di presenza con molti NAS e tutti devono condividere le stesse informazioni sugli utenti. Il server di autenticazione ha accesso a un database che contiene le credenziali dell'utente e la configurazione per ogni utente (a seconda del contratto tra utente e ISP). Il server di autenticazione risponde al NAS con la risposta (utente valido/invalido) e la configurazione che il NAS deve applicare al traffico dell'utente.

I produttori di NAS sostengono che la sicurezza necessita di tre funzioni, brevemente denominate **AAA**:

- **Autenticazione**: l'identità dell'entità viene autenticata in base alle credenziali (ad esempio, password, OTP);
- **Autorizzazione**: determinare se un'entità è autorizzata a svolgere una determinata attività o ad accedere alle risorse o ai servizi;
- **Contabilità**: tracciamento dell'uso delle risorse di rete per supporto alla revisione, analisi della capacità o fatturazione dei costi. Il SA svolge esattamente queste tre funzioni dialogando con uno o più NAS tramite uno o più protocolli.

Protocolli di autenticazione di rete

Fondamentalmente, esistono tre protocolli che il SA e il NAS possono utilizzare per comunicare:

- **RADIUS**: è lo standard de-facto (e il più usato) con una bella caratteristica che gli permette di funzionare come un proxy verso altri sistemi di autenticazione. Può essere sia un sistema di autenticazione che utilizzare AS esterni;
- **DIAMETER**: è l'evoluzione di RADIUS e pone l'accento sul roaming tra diversi ISP e, essendo più moderno, si è occupato maggiormente della sicurezza;
- **TACACS+ (TACACS, XTACCS)**: concorrente di Radius, tecnicamente migliore ma che ha ottenuto una minore accettazione perché era una soluzione proprietaria, implementata solo da Cisco senza alcuna specifica pubblica.

RADIUS

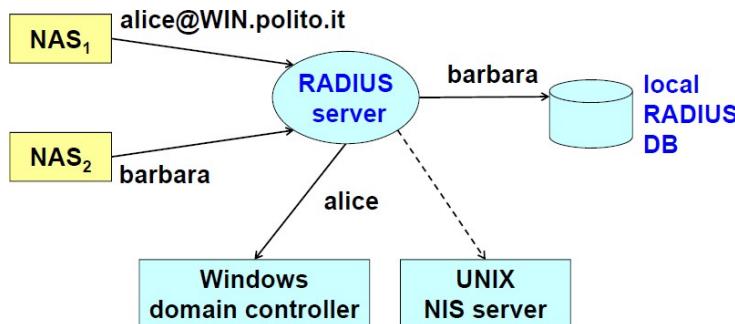
Radius è l'acronimo di **Remote Authentication Dial-In User Service**. Si tratta di un protocollo molto vecchio, perché è stato considerato per il Dial-In (quando gli utenti componevano il numero per connettersi a un ISP con il modem), ed è stato inventato quasi 30 anni fa. Si è evoluto nel tempo e **supporta l'autenticazione, l'autorizzazione e la contabilità** per controllare l'accesso alla rete:

- *Porte fisiche* (analogiche, ISDN, IEEE 802): RADIUS può essere utilizzato anche per eseguire il controllo degli accessi alla rete Ethernet.
- *Porte virtuali* (tunnel, accesso wireless): per gestire una rete di punti di accesso in modo centralizzato.

RADIUS come concetto implementa l'**amministrazione e la contabilità centralizzate** (per memorizzare tutte le informazioni sull'uso delle risorse). È un protocollo **client-server** tra il **NAS** e il **SA** e utilizza la **porta 1812/UDP** (autenticazione) e la **porta 1813/UDP** (contabilità). Poiché UDP non è affidabile, ogni trasmissione di un pacchetto RADIUS è soggetta a timeout. Se non viene ricevuto alcun ACK dopo il timeout, lo stesso pacchetto viene ritrasmesso e c'è un numero massimo di appends dopo il quale la comunicazione viene dichiarata impossibile. RADIUS supporta architetture con un server principale ma anche molti **server secondari**: ciò è utile per migliorare le prestazioni e la resistenza del sistema (anche agli attacchi DoS).

- RFC-2865 (Protocollo)
- RFC-2866 (contabilità)
- RFC-2867/2868 (contabilità e attributi dei tunnel)
- RFC-2869 (estensioni)
- [RFC-3579 \(supporto RADIUS per EAP\)](#)
- RFC-3580 (linee guida per 802.1X con RADIUS)

Dall'elenco sopra riportato è possibile notare che dal protocollo di base sono state apportate numerose estensioni (anche il supporto EAP) e in particolare supporta anche [802.1X](#), un'architettura di sicurezza per il controllo degli accessi alla rete.



Il **server RADIUS** può agire come proxy verso altri server di autenticazione. Osservando l'immagine, a sinistra ci sono due NAS, ognuno dei quali fornisce una richiesta di accesso da parte degli utenti. RADIUS consulta il suo *DB RADIUS locale* sulla destra e controlla se (ad esempio, Barbara) è un utente registrato valido. Nel caso in cui riceva una richiesta di accesso da un altro NAS, magari sotto forma di indirizzo e-mail, il RADIUS controlla se (ad esempio, Barbara) è un utente registrato valido.

(anche se non è un indirizzo e-mail) *alice@WIN.polito.it* indica che l'utente Alice non è registrato nel DB RADIUS locale, ma che *Alice* è un utente definito nel dominio di sicurezza *WIN.polito.it* a cui il server RADIUS è in qualche modo associato. Ciò significa **che RADIUS agirà come proxy per la parte di autenticazione** e reindirizzerà la richiesta al *controller di dominio Windows*. Quindi l'autorizzazione/contabilità potrebbe essere gestita localmente dal server RADIUS. RADIUS può anche essere associato a un altro dominio (ad esempio, un *server NIS UNIX*).

Quali funzionalità di sicurezza per Radius?

Radius necessita di funzionalità di sicurezza a causa di:

- **Sniffing delle richieste NAS (se contiene password):**

- Problema di *riservatezza*, nel senso che se la richiesta contenesse la password in chiaro sarebbe facilmente copiabile. Anche se la password non viene inviata (ad esempio, sistema OTP) esiste un problema di *privacy*, che è diverso da quello della riservatezza, perché se qualcuno sniffasse il traffico saprebbe che qualcuno sta eseguendo delle azioni (ad esempio, "Lioy è

connesso alla rete"). Non ci sono problemi di sniffing della risposta (dice solo al NAS se è valida o meno).

- **Risposta AS falsa (per bloccare l'utente valido o consentire quello non valido):**
 - Questo potrebbe essere utilizzato perché, trattandosi di UDP, si potrebbe essere più veloci nella risposta rispetto al server reale. La prima risposta ricevuta è considerata valida, quindi la risposta falsa potrebbe essere usata per DoS (per bloccare un utente valido) o dare una risposta positiva anche se l'utente non è valido. In questo caso è necessaria l'autenticazione della risposta.
- **Modifica della risposta del SA ($Y \rightarrow N$ o $N \rightarrow Y$):**
 - Potrebbe anche essere possibile modificare una risposta da valida a non valida e viceversa. Per questo motivo, sono necessarie l'autenticazione e l'integrità delle risposte; l'integrità della risposta è necessaria per evitare questo tipo di attacco.
- **Riproduzione della risposta del SA (se non adeguatamente legata alla richiesta del NAS):**
 - Una risposta può essere riprodotta se non è legata a una richiesta specifica. Se la risposta è solo "valida" o "non valida", è possibile prendere quella valida e riprodurla a un altro utente. Per evitare questo tipo di attacco, la risposta dovrebbe contenere l'identificazione dell'utente (ad esempio, "Antonio Lioy che si è collegato alle ore XX:XX è valido").
- **Enumerazione delle password (da un falso NAS):**
 - Se qualcuno riesce a connettersi alla rete back-end tra il NAS e il server Radius, potrebbe creare un falso NAS e iniziare a inviare richieste al server Radius. Per evitare ciò, è necessaria l'autenticazione delle richieste del NAS.
- **DoS (molte richieste di NAS da NAS falsi):**
 - Se è possibile collegare alla rete di back-end un falso NAS, è possibile inondare il server Radius con molte richieste, fino a renderlo non disponibile. I NAS sono configurati con un elenco di vari server Radius. Per questo motivo, se il tempo scade e non arriva alcuna risposta, il NAS presume che il server sia occupato e passa a quello successivo. La resistenza all'attacco è proporzionale al numero di server secondari che è possibile configurare nel sistema.

Caratteristiche di RADIUS

Per la **protezione dei dati** sono disponibili l'integrità dei pacchetti e l'autenticazione tramite **keyed-MD5**:

- La chiave è un segreto condiviso (tra il NAS e il server Radius): l'inserimento di un falso server NAS nella rete comporterà il rifiuto delle sue richieste;
- nel caso in cui i pacchetti contengano una password (ad esempio, l'utente ha deciso di usare PAP e la password è stata inviata in chiaro), allora la password viene trasmessa "criptata" (non una vera e propria crittografia, ma solo offuscata usando MD5, che non è un algoritmo di crittografia ma solo di digest): $\text{password} \oplus \text{md5}(\text{key} + \text{authenticator})$. Questo viene imbottito con byte NULL fino a raggiungere un multiplo di 128 bit.

Radius supporta tutti i tipi di autenticazione (PAP, CHAP, token-card ed EAP); Cisco fornisce un server gratuito per CryptoCard; altri supportano anche SecurID. L'aspetto importante è che Radius è un **protocollo estensibile**, perché ogni pacchetto trasporta alcuni attributi descritti nella forma **TLV (tipo di attributo - lunghezza - valore)**. Ciò consente l'introduzione di nuovi tipi senza interrompere la compatibilità.

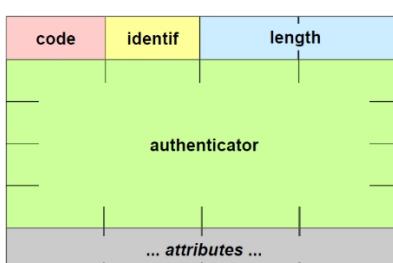


Figura 46 Formato di un pacchetto RADIUS

Il formato di un pacchetto Radius è il seguente:

- Codice (8 bit);
- Identificatore (8 bit);
- Lunghezza (16 bit);
- Autenticatore (128 bit);
- Elenco TLV di attributi.

Esistono molti tipi di pacchetti, alcuni dei quali sono:

- **ACCESS-REQUEST**: contiene le credenziali di accesso (ad esempio, nome utente e password);
- **ACCESS-REJECT**: l'accesso è negato (ad esempio, a causa di username/password errati);

- **ACCESS-CHALLENGE**: richiede all'utente informazioni aggiuntive (ad esempio, PIN, codice token, password secondaria);
- **ACCESS-ACCEPT (parametri)**: l'accesso è consentito e vengono forniti alcuni parametri di rete (ad esempio, IPAddr, netmask, MTU, host, porta, ...).

All'interno dei pacchetti è presente **l'autenticatore**, che ha un doppio scopo: nella *risposta del server* fornisce *l'autenticazione e la protezione dal replay; maschera la password*. In Access-Request è denominato *Request Authenticator* ed è costituito da 16 byte generati casualmente dal NAS. Nelle risposte del server, oltre a chiamarsi Response Authenticator, non è casuale, ma viene calcolato tramite un *digest con chiave*:

$$MD5(\text{codice} \parallel \text{ID} \parallel \text{lunghetza} \parallel \text{RequestAuth} \parallel \text{attributi} \parallel \text{segreto})$$

Si noti che la presenza di *RequestAuth* nel calcolo rende una risposta legata a una richiesta, per cui non è possibile eseguire attacchi di replay.

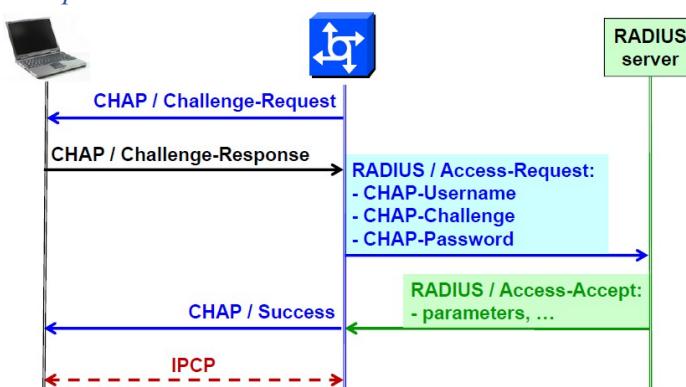
Gli attributi hanno forma di TLV e alcuni di essi sono:

type	length	value
------	--------	-------

- **Tipo = 1 (nome utente)**
 - Il valore può essere una stringa di testo, un identificatore di accesso alla rete (NAI) o un DN (Distinguished name).
- **Tipo = 2 (password utente)**
 - Il valore è la password \oplus md5 (key || RequestAuth)
- **Tipo = 3 (parola chiave)**
 - Il valore è la risposta CHAP dell'utente (128 bit).
- **Tipo = 60 (CHAP-Challenge)**
 - Il valore è la sfida lanciata dal NAS all'utente.

Il **NAI (Network Access Identifier)** viene utilizzato per distinguere se la richiesta proviene da un utente locale o da un utente appartenente a un dominio di sicurezza diverso. Il NAI è sotto forma di nome utente e, facoltativamente, di dominio di sicurezza. Le regole dicono che tutti i dispositivi devono supportare NAI fino a 72 byte. La sintassi esatta per username e realm è descritta in dettaglio nella RFC, ma sono ammessi solo caratteri ASCII < 128, ma tutti (anche quelli non stampabili). Il nome utente è quello usato nella fase di autenticazione PPP, cioè quello usato quando si apre la connessione al livello che non corrisponde necessariamente al nome utente dell'applicazione.

Esempio: CHAP + RADIUS



La CHAP viene utilizzata per l'autenticazione e Radius per verificare l'autenticazione. A sinistra c'è un utente, al centro il NAS e a destra il server Radius.

Quando l'utente si connette al sistema, il NAS invia un pacchetto CHAP contenente una *richiesta di sfida*. Il client inserisce la password e fornisce la *risposta di sfida*. Si noti che il NAS non dispone della password e non può verificare se la risposta è valida o meno. Per questo motivo, creerà un pacchetto *Richiesta di accesso Radius* con tutte le informazioni necessarie.

informazioni: *CHAP-Username* è quello fornito dall'utente, *CHAP-Challenge* è la sfida inviata dal NAS all'utente e *CHAP-Password* è la risposta alla sfida. Con queste informazioni il server Radius può verificare se la risposta alla sfida è buona o meno. Immaginando che la risposta sia buona, il server Radius invierà una risposta *Radius Access Accept* con i parametri di rete per l'utente specifico. Questo verrà tradotto in CHAP sotto forma di pacchetto *Success* e a quel punto l'L3 verrà abilitato (ad esempio, IPCP). Il NAS sta creando

un dialogo con l'utente disposto ad accedere alla rete e sta trasformando le informazioni da un protocollo a un altro.

Radius presuppone che ci si trovi all'interno del sistema di accesso alla rete di un unico provider.

DIAMETRO

Diameter è un'evoluzione di RADIUS che pone particolare enfasi sul roaming tra ISP. Quando si visitano infrastrutture diverse, DIAMETER consente l'autenticazione a un altro sistema. Sono state definite molte RFC:

- *RFC-3588 "Protocollo di base Diameter"*
- ***RFC-3589 "Comandi per il 3GPP"*** (antenato di 3G, 4G, 5G)
- *RFC-3539 "Profilo di trasporto AAA"*
- *RFC-4004 "Applicazione Diameter mobile IPv4"*
- *RFC-4005 "Applicazione server di accesso alla rete Diameter".*
- *RFC-4006 "Applicazione del diametro per il controllo del credito".*
- *RFC-4072 "Applicazione Diameter EAP"*

Diameter ha una **sicurezza migliore di Radius**: fornisce una protezione con un meccanismo esterno su tutti i pacchetti scambiati. Invece di inserire la sicurezza all'interno del protocollo, i pacchetti Diameter devono essere trasmessi all'interno di un canale di rete sicuro (come IPsec e TLS). È obbligatorio che Diameter client supporti almeno IPsec e, facoltativamente, anche TLS. Al contrario, Diameter Server deve supportare entrambi i protocolli di sicurezza.

Dal punto di vista delle configurazioni, IPsec ha un formato denominato *ESP* che fornisce la crittografia. IPsec deve essere configurato con un **algoritmo non nullo** sia per l'autenticazione che per la privacy. Nel caso di TLS, deve supportare l'**autenticazione reciproca**, che richiede che il client disponga di un certificato a chiave pubblica.

TLS ha come requisiti minimi il supporto di RSA per l'autenticazione, quindi RC4_128/3DES e MD5/SHA1 per l'integrità. Opzionalmente, può supportare RSA con AES_128 e SHA1.

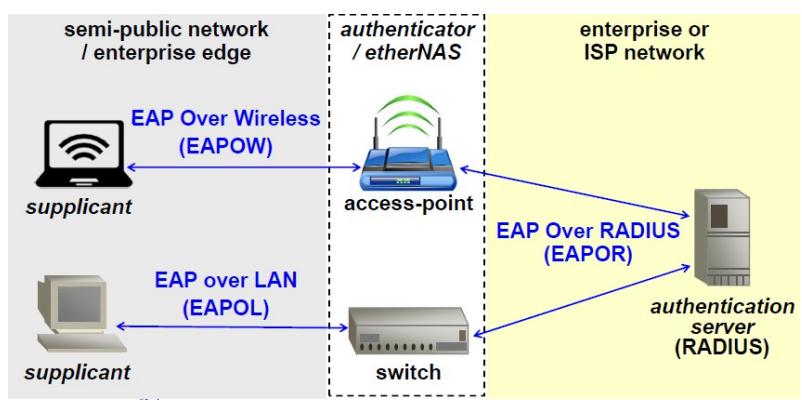
Radius e Diameter sono entrambi protocolli che consentono la centralizzazione del controllo degli accessi.

IEEE 802.1x

Radius e Diameter sono stati utilizzati per definire IEEE 802.1x, un'architettura più generale nota anche come **Port-Based Network Access Control**. È un'architettura di autenticazione che funziona su L2. Verifica l'identità e l'autorizzazione degli utenti prima di lasciarli comunicare. Può essere *utile* in una rete cablata per bloccare un accesso non autorizzato, ma è **assolutamente necessario nelle reti wireless**. Le prime implementazioni sono state immediatamente realizzate da Windows-XP e dagli access-point wireless Cisco.

IEEE 802.1x è un framework, il che significa che non impone una soluzione di autenticazione specifica, ma ne supporta molte. In particolare, è un framework per eseguire l'**autenticazione** e la **gestione delle chiavi**. Il secondo punto è necessario per le reti wireless, perché la parte radio di una comunicazione wireless può essere facilmente sniffata. Con la gestione delle chiavi è possibile definire una chiave simmetrica condivisa che può essere utilizzata per criptare il traffico. Può derivare chiavi di sessione da utilizzare per l'autenticazione, l'integrità e la riservatezza dei pacchetti. Utilizza algoritmi standard per la derivazione delle chiavi (ad esempio, TLS, SRP, ...) e dispone di servizi di sicurezza opzionali (autenticazione o autenticazione e crittografia).

A sinistra, l'immagine mostra la **rete semi-pubblica** (o *enterprise edge*), cioè la rete che contiene i dispositivi che vogliono accedere alla rete, chiamati **supplicant**. L'elemento che si trova all'interfaccia tra la rete centrale e l'edge prende il nome di **authenticator** o **etherNAS** (può essere un *access point* o uno *switch*) e, per i supplicant, è possibile connettersi in più modi ad



esso. In particolare,

802.1x utilizza *EAP* per eseguire l'autenticazione e, a seconda che venga utilizzato su una rete wireless o cablata, prende il nome rispettivamente di *EAP Over Wireless (EAPOW)* o *EAP Over LAN (EAPOL)*. In ogni caso, quando

L'autenticatore riceve la richiesta EAP da un supplicant e verifica se è valida o meno eseguendo la decapsulazione e la ricapsulazione del pacchetto in un altro protocollo (Radius). Sul retro c'è *EAP Over Radius* (**EAPOR**) per verificare se l'utente è valido o meno.

802.1x - vantaggi

Sfrutta il livello applicativo per l'effettiva implementazione dei meccanismi di sicurezza. Esiste un dialogo diretto tra supplicant e AS (server di autenticazione), quindi i dispositivi utente parlano direttamente con il server Radius. La scheda di rete (NIC) e il NAS funzionano come "*dispositivo passante*", ossia si limitano all'incapsulamento e al decapsulamento. Questo è importante perché non sono necessarie modifiche (a livello di NIC e NAS) per implementare nuovi meccanismi di autenticazione. Questi meccanismi devono essere implementati solo su Radius Server e Supplicant. Questo è importante perché questa architettura di sicurezza non deve cambiare anche se in futuro ci sarà un'evoluzione delle tecniche di autenticazione. Infine, poiché utilizza Radius, questo sistema si integra perfettamente nelle architetture AAA che consentono anche l'accounting.

802.1x - Messaggi

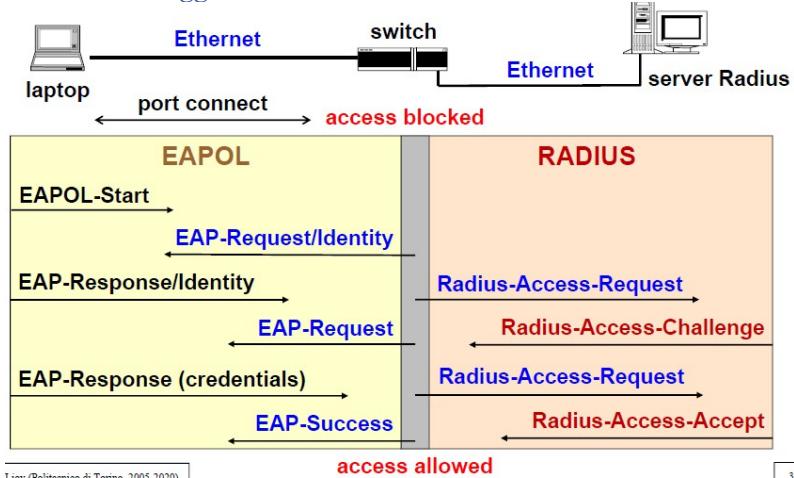


Figura 47 Autenticazione di un supplicant utilizzando l'architettura 802.1x: lo switch è un dispositivo pass-through, traduce i messaggi EAP in RADIUS e viceversa, ma la connessione è end-to-end

Nell'esempio, il portatile è collegato via Ethernet a uno switch, in cui per impostazione predefinita tutte le porte Ethernet sono bloccate. Quindi, il supplicant avvia una negoziazione tramite *EAPOL-Start*. La risposta dello switch è la *richiesta/identità EAP* e il supplicant fornisce con la risposta/identità EAP la sua identità, che viene inoltrata al server Radius (*richiesta di accesso Radius*). In questo caso lo switch agisce come un elemento di passaggio. A questo punto, il server Radius risponde con una sfida tramite il pacchetto *Radius-Access-Challenge*, che viene poi tradotto in un pacchetto *EAP-Request* dallo switch. Il supplicante, ancora una volta, fornirà il pacchetto

risposta alla sfida che verrà tradotta in un'altra richiesta di accesso Radius. Quindi, il server Radius accetterà l'utente e lo scambio si concluderà con un pacchetto *EAP-Success* inviato all'utente. L'utente sarà ora in grado di comunicare con le reti L3 e superiori.

Esempio di Eduroam

Un esempio importante di utilizzo di Radius è la rete **Eduroam**. Si tratta di una rete mondiale di controllo degli accessi che coinvolge tutte le università e molti centri di ricerca in tutto il mondo. Circa un anno fa sono stati collegati 101 Paesi e altri 18 Paesi sono in fase di test. Utilizza 802.1x e la federazione Radius.

Il Supplicant può vedere che c'è un access point in un'università e può connettersi ad esso, perché la connessione si chiama Eduroam. Il punto di accesso si riferisce all'AS locale, che si chiama **visit AS**. Poiché utilizzerà la sintassi Network Access Identifier (NAI), il supplicante utilizzerà il suo identificatore (*ad esempio, s123456@studenti.polito.it*) e il Radius Server locale saprà che deve attraversare la **gerarchia Eduroam** (nazionale, internazionale...) fino a raggiungere il Radius AS in cui il supplicante ha creato le sue credenziali (*ad esempio, il PoliTO Radius Server*), che è chiamato **Home AS**. Una volta trovato, ci sarà una connessione diretta attraverso un canale virtuale sicuro E2E (*End-to-end*) (*ad esempio, EAP-TTLS*) tra il supplicant e l'Home AS per eseguire l'Autenticazione e quest'ultimo fornirà la risposta all'access point, che permetterà all'utente di navigare.

Implementazione della sicurezza nei livelli OSI

Application
Presentation
Session
Transport
Network
Data Link
Physical



La domanda principale è "Qual è il miglior livello OSI per implementare la sicurezza?" con molte possibilità e risposte. In genere, "Presentazione" è l'unico in cui le misure di sicurezza non sono utili.

Purtroppo non esiste un unico livello ottimale. Più si sale nello stack, più specifica può essere la funzione di sicurezza. Ad esempio, a livello di applicazione è possibile identificare l'utente, i comandi e i dati. Le funzioni di sicurezza sono anche indipendenti dalla rete sottostante, ma, se le funzioni sono collocate a livello di

solo a livello di applicazione, sono possibili attacchi a livelli inferiori (in particolare, sono disponibili attacchi DoS).

Più si scende nella pila, più rapidamente si possono "espellere" gli intrusi, ma meno sono i dati per la decisione (ad esempio, solo gli indirizzi MAC o IP, nessuna identificazione dell'utente, nessun comando).

In breve: non esiste un livello ottimale. Si può decidere se correre uno di questi due rischi o fare un mix, collocando alcune caratteristiche di sicurezza a livelli inferiori e concentrando la maggior parte della sicurezza a livello di applicazione.

Quando si raggiunge L3, una delle prime cose attivate è il DHCP, perché l'accesso alla rete è stato dato e ora l'utente deve conoscere i parametri di rete. Il DHCP è il protocollo con cui un dispositivo può chiedere l'assegnazione di un indirizzo di rete valido. Sfortunatamente, il protocollo **non è autenticato** ed è un protocollo **broadcast** che fornisce una risposta contenente *indirizzo IP, netmask, gateway predefinito, nameserver locale e suffisso DNS locale*. Per questo motivo, l'attivazione di un falso server DHCP è banale, perché la richiesta DHCP è un frame broadcast L2 e l'unica cosa che un aggressore deve fare è rimanere nello stesso dominio broadcast della vittima e sniffare la richiesta DHCP.

Sicurezza (in)DHCP

I possibili attacchi del falso DHCP sono:

- **Negazione del servizio**
 - Ciò può avvenire fornendo una configurazione di rete errata;
- **(logico) MITM**
 - Alla vittima viene fornito un indirizzo IP valido, ma gli viene assegnata una sottorete con solo gli ultimi due bit uguali a zero. Pertanto, solo due indirizzi sono validi: uno viene assegnato all'utente e l'altro all'aggressore come gateway predefinito. In questo modo, la macchina attaccata è isolata in una propria sottorete (logicamente, non fisicamente). Per comunicare con tutti i nodi del mondo, la vittima deve inviare tutto attraverso l'attaccante;
 - Le risposte potrebbero arrivare al nodo originale senza passare attraverso l'attaccante. Per questo motivo, è possibile attivare il NAT e intercettare anche le risposte.
- **Traduzione malevola di nome e indirizzo**
 - L'aggressore si dichiara come server dei nomi locale. Quindi, ogni volta che l'utente deve eseguire una traduzione da nome a indirizzo, l'aggressore fornirà l'indirizzo sbagliato. Questo viene utilizzato, ad esempio, per il phishing e il pharming.

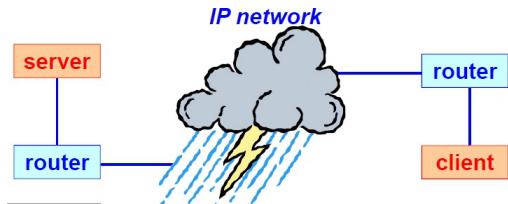
Diversi produttori hanno cercato di migliorare la sicurezza, come ad esempio gli switch (ad esempio, Cisco) che offrono:

- **DHCP snooping**: accetta solo risposte da "porte fidate";
- **IP guard**: offre spazio solo agli indirizzi IP ottenuti da un server DHCP valido (ma c'è un limite al numero di indirizzi riconosciuti).

Esiste anche l'RFC-3118 "*Authentication for DHCP messages*" che utilizza *HMAC-MD5* per autenticare i messaggi, ma è raramente adottato perché difficilmente configurabile: essendo HMAC un protocollo simmetrico, è

necessario installare una chiave su tutti i computer che devono utilizzare il DHCP. Ciò comporta il problema della distribuzione delle chiavi. Inoltre, c'è un problema di gestione delle chiavi, perché se una chiave viene acquisita sarà riutilizzabile e, poiché è simmetrica, qualsiasi client DHCP potrebbe funzionare anche come server DHCP.

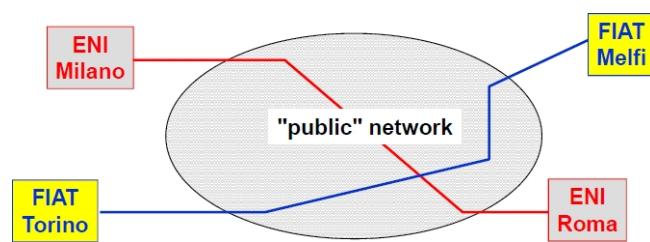
Sicurezza a livello di rete (L3)



Il livello 3, che, dato l'uso odierno di Internet è l'IP, è un livello in cui è possibile creare caratteristiche di sicurezza significative in senso generale, perché è il primo livello che fornisce connettività end-to-end. Ciò consentirebbe di creare una **protezione end-to-end** per reti L3 omogenee (come le reti IP) e **VPN (Virtual Private Network)**.

Se è possibile fornire una protezione end-to-end L3, in modo che i dati siano protetti non appena escono dal server/client e non importa se i router non sono gestiti correttamente o se la rete che si sta attraversando non è sicura, perché i dati sono protetti non appena escono dall'interfaccia di rete, fino a quando non raggiungono l'interfaccia di rete finale: quindi, gli unici attacchi possibili sono quelli interni al client/server. Quindi, la sicurezza a livello L3 permette di dimenticare tutti gli altri attacchi a livello di rete (*a parte il DoS*).

Che cos'è una VPN?



La Virtual Private Network è una tecnica (hardware e/o software) che permette di creare una rete privata pur utilizzando canali e dispositivi di trasmissione condivisi (o comunque non affidabili). Piuttosto che posare i propri cavi e gestirli in proprio, ci possono essere aziende che desiderano avere un segmento virtuale della rete. Per

Ad esempio, è possibile dire che i pacchetti FIAT sono etichettati come blu e possono essere scambiati solo tra end point blu, mentre per ENI sono rossi e possono essere scambiati solo attraverso end point rossi. È un buon concetto, ma il diavolo si nasconde nei dettagli: poiché non è possibile immaginare i pacchetti blu o rossi e non è possibile essere sicuri di come vengono scambiati, i dettagli dell'implementazione sono importanti.

Esistono tre tecniche per creare una VPN:

- **Indirizzamento privato**
- **Routing protetto (tunnel IP)**
- **Protezione crittografica dei pacchetti di rete (tunnel IP sicuro)**

VPN tramite indirizzi privati

In questa implementazione VPN di base, le reti che fanno parte della VPN utilizzano indirizzi non pubblici in modo da essere irraggiungibili da altre reti (ad esempio, reti private IANA come quelle elencate nella RFC-1918). Pertanto, queste reti sono private nel senso che non richiedono un'autorizzazione e i pacchetti in questo caso non sono instradabili a livello globale. Ad esempio, un provider di telecomunicazioni che voglia condividere la propria infrastruttura con diversi clienti potrebbe assegnare una classe di indirizzi diversi a ciascun cliente e poi potrebbe utilizzare liste di controllo degli accessi sui router per costringere i pacchetti ad andare solo verso le destinazioni consentite.

In molti casi, questa protezione può essere facilmente violata da qualcuno:

- *Indovina o scopre gli indirizzi*
Se una classe di indirizzi viene utilizzata da un altro cliente e qualcuno scopre questa classe di indirizzi, potrebbe cambiare il proprio indirizzo per infiltrarsi nella rete.

- *Può sniffare i pacchetti durante la trasmissione*
Poiché i pacchetti non hanno alcuna protezione intrinseca, se è possibile sniffare la rete, può essere possibile anche leggere il contenuto dei pacchetti.
- *Ha accesso ai dispositivi di comunicazione*

È possibile leggere/modificare/iniettare qualsiasi tipo di pacchetto.

In questo caso non c'è una vera protezione per i pacchetti, per i clienti e nemmeno per il gestore dell'infrastruttura. Pertanto, il livello reale di sicurezza è prossimo allo zero, anche se questo tipo di servizio è offerto commercialmente.

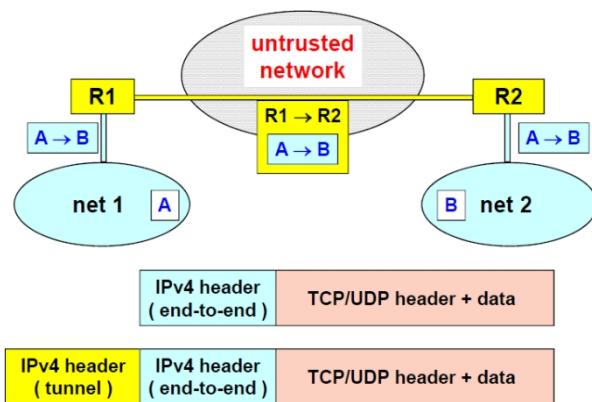
VPN tramite tunnel

Questa soluzione è leggermente migliore della precedente. In questo caso i router encapsulano l'intero pacchetto L3 come *payload* all'interno di un altro pacchetto. Può essere **IP in IP**, **IP su MPLS** o altre tecniche. Prima di procedere all'incapsulamento, i router di confine eseguono il controllo dell'accesso alla VPN tramite **ACL** (Access Control List). Ad esempio, se una rete appartiene all'indirizzo *10.1*, la destinazione può essere solo un'altra persona in *10.1*.

Con questa soluzione, i provider ottengono una protezione contro gli utenti finali malintenzionati perché evita che i clienti possano cambiare la sottorete di appartenenza. Tuttavia, questa protezione può essere sconfitta da *chiunque gestisca un router* o possa *sniffare* i pacchetti durante la trasmissione, ad esempio non offre protezione ai clienti contro gli attacchi che possono essere eseguiti dall'interno della rete geografica (→ protezione per i provider ma non per i clienti).

Se vogliamo davvero una protezione, dobbiamo passare ad altre tecniche.

VPN tramite tunnel IP



La rete 1 e la rete 2 sono dello stesso colore perché appartengono alla stessa sottorete. Se si utilizza un tunnel IP, quando i pacchetti vanno dal nodo A della sottorete 1 al nodo B della sottorete 2, raggiungono i router di confine della sottorete 1 che hanno il compito di incapsularli.

Il router R1 saprà che B si trova nella sottorete 2, raggiungibile attraverso il router di confine R2, e creerà un altro pacchetto che va da R1 a R2 e che contiene come payload il pacchetto originale.

Nell'immagine è mostrata l'intestazione IPv4 esterna del tunnel. Quando il pacchetto viene ricevuto dal router R2, viene decapsulato e inviato alla destinazione finale. **Durante la trasmissione, il pacchetto può essere letto, manipolato, iniettato** (ancora una volta, nessuna sicurezza reale per l'utente finale della VPN).

Il tunnel IP presenta anche un problema di prestazioni: la **frammentazione**. Se il pacchetto ha dimensioni pari alla MTU, l'incapsulamento sarà possibile solo con la frammentazione. In questo caso, la perdita massima di prestazioni è pari al 50%, perché vengono generati due pacchetti anziché uno. La perdita maggiore si ha per le applicazioni con pacchetti di grandi dimensioni (tipicamente le applicazioni non interattive, ad esempio il trasferimento di file). Ciò significa che questa soluzione può anche essere un killer delle prestazioni.

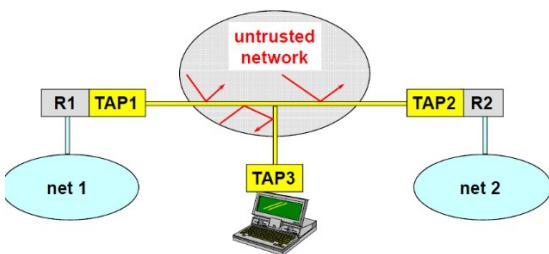
VPN tramite tunnel IP sicuro

Per il problema del performance killer non c'è soluzione ma, con quest'ultima soluzione, si può fare qualcosa per gli utenti finali. Prima dell'incapsulamento, i pacchetti vengono protetti con:

- **MAC (integrità e autenticazione)**
- **Crittografia (riservatezza)**
- **Numerazione (per evitare attacchi di replay)**

Non esiste la firma digitale, perché è **molto lenta** e non sarebbe mai in grado di adattarsi alla velocità delle reti attuali. Se gli algoritmi crittografici selezionati sono forti, l'unico attacco possibile è quello di bloccare le

comunicazioni (*DoS*). A volte, questo tipo di VPN è noto anche come **S-VPN** (*Secure VPN*). Questa è l'unica VPN sicura (→ attenzione alle VPN sponsorizzate online)!



gestito dall'ISP.

Nell'immagine sono presenti un router e un **TAP** (*Tunnel Access Point*). Le funzioni sono divise: il router supervisiona l'incapsulamento/decapsulamento, mentre il TAP è quello che esegue la protezione crittografica. Se si utilizza questa soluzione e il TAP viene dato in gestione a un fornitore di rete esterno, si tratta di una **falsa sicurezza**. Perché ci dovrebbero essere due dispositivi separati: il TAP dovrebbe essere gestito dal client e il router dovrebbe essere gestito dall'ISP.

IPsec

IPsec è l'architettura IETF per la sicurezza L3 in IPv4/IPv6 per **creare S-VPN su reti non fidate** e per **creare flussi di pacchetti sicuri end-to-end**. Ciò è possibile grazie alla definizione di due tipi di pacchetti specifici:

- **AH (Authentication Header):**
Per garantire *l'integrità, l'autenticazione e la protezione contro gli attacchi replay*.
- **ESP (Encapsulating Security Payload):**
Fornisce *quasi le stesse funzioni di AH*, con in più la *riservatezza (del carico utile)*.

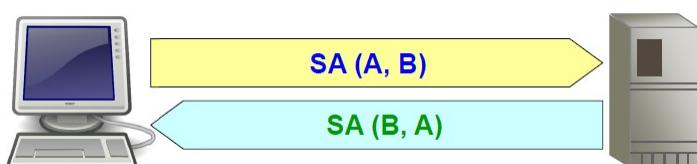
È importante sottolineare che la **riservatezza può essere garantita solo per il payload**: non è mai possibile criptare l'header, altrimenti i sistemi intermedi non sarebbero in grado di elaborare i pacchetti.

Esiste anche un protocollo dedicato allo scambio di chiavi, denominato **IKE (Internet Key Exchange)**, per creare e distribuire chiavi nelle reti IP.

I servizi di sicurezza IPsec sono:

- **Autenticazione dei pacchetti IP:**
 - *Integrità dei dati*: il ricevitore può **rilevare** se il pacchetto è stato manipolato. Non ha lo scopo di evitare la manipolazione;
 - *Autenticazione del mittente*, ovvero una prova formale dell'identità del mittente. Si noti che non corrisponde a un indirizzo IP: Non bisogna fidarsi degli indirizzi IP, perché possono essere completamente falsi (*IP spoofing*);
 - *Protezione (parziale) contro gli attacchi "replay"*: ci sono problemi legati al fatto che stiamo lavorando a livello L3 e i pacchetti possono essere persi o duplicati;
- **Riservatezza dei pacchetti IP:**
 - *Cifratura dei dati* (solo per il carico utile);

Associazione di sicurezza (SA) IPsec



Queste caratteristiche di sicurezza sono associate al concetto di **Security Association (SA)**, che è una **connessione logica unidirezionale tra due sistemi IPsec**. A ogni SA sono associati diversi servizi di sicurezza. Per ottenere una completa

per un flusso di pacchetti bidirezionale tra due nodi **sono necessarie due SA** (una da A a B e una per i pacchetti da B ad A). Questo significa che, in teoria, è possibile avere caratteristiche di sicurezza diverse e algoritmi diversi per le due direzioni, ma di solito anche se ci sono due SA distinte viene utilizzato lo stesso tipo di protezione/algoritmi.

Immaginiamo che il mittente (nodo A) stia trasmettendo dei dati (il che significa che potrebbe volere la riservatezza), ma che la risposta di B sia una cosa molto semplice (ad esempio "ricevuto" o "cattivo") che non necessita di riservatezza: allora è possibile evitare la crittografia del pacchetto di ritorno.

Le associazioni di sicurezza sono gestite attraverso due database locali, che **non sono veri e propri database** (*cioè* non sono implementati server come SQL o Oracle, ma sono solo una *raccolta di dati*):

- **SPD (Database dei criteri di sicurezza):**
 - Contiene un elenco di criteri di sicurezza da applicare ai diversi flussi di pacchetti;
 - configurato a priori (ad esempio manualmente) o collegato a un sistema automatico (ad esempio ISPS, che sta per Internet Security Policy System);
- **SAD (Database SA):**
 - È un database di runtime che contiene l'elenco delle SA attive e le loro caratteristiche (*ad esempio, algoritmi, chiavi, parametri*) per creare traffico protetto per quella specifica SA;

Supponiamo di trovarci in un nodo di invio all'interno dello stack TCP/IP. È stato creato un pacchetto IP da inviare a L2, ma su questo nodo è presente IPsec. Quando il pacchetto è pronto per essere inviato, il modulo IPsec inizia a funzionare. La prima domanda è: *quale politica deve essere applicata a questo pacchetto?* La risposta viene fornita dall'**SPD**. Può essere "*dovresti applicare queste regole di sicurezza*" oppure "*non hai bisogno di alcun tipo di protezione per questo pacchetto. Passare direttamente a L2*". Se la protezione è necessaria e questo è il primo pacchetto di questo specifico flusso di rete, IPsec procede alla creazione di un'associazione di sicurezza, altrimenti esiste già un'associazione di sicurezza.

SA esistente e procede alla lettura del parametro associato a quel SA consultando il **SAD**. Questo gli fornirà gli algoritmi e i parametri per arricchire il e, infine, il pacchetto IP sarà protetto con IPsec e inviato a L2 per la trasmissione effettiva.

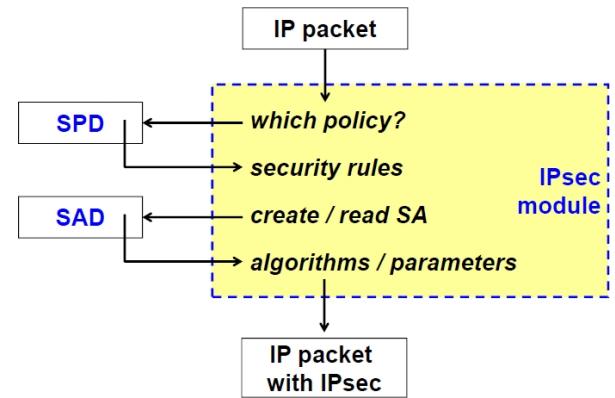


Figura 48 Operazioni logiche eseguite da un modulo IPsec durante l'invio di un pacchetto

0	4	8	16	19	31					
vers.	IHL	TOS	total length							
identification		flags	fragment offset							
TTL	protocol	header checksum								
source IP address										
destination IP address										
options			padding							

Figura 49 Intestazione IPv4, come promemoria dei vari campi: il valore di alcuni campi cambia durante l'inoltro del pacchetto, e questo deve essere tenuto in considerazione nel calcolo dell'hash.

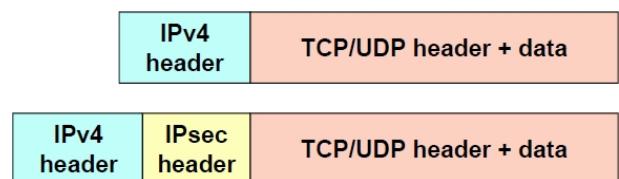
Nell'immagine c'è l'intestazione IPv4. Poiché la riservatezza può essere applicata solo al carico utile, è possibile proteggere qualcosa dell'intestazione? Potrebbe essere possibile fornire autenticazione e integrità. In genere questi risultati richiedono il calcolo di un hash, che presuppone che i dati non cambino. Purtroppo, durante la trasmissione alcuni di questi parametri cambiano, come il TTL e il checksum. In genere, il NAT non può essere utilizzato con IPsec, perché cambiando l'indirizzo IP di origine, cambia anche l'hash e quindi il controllo eseguito a destinazione fallisce. Ci sono anche alcune opzioni (ad esempio, il routing della sorgente, cioè il numero di router da attraversare) che cambiano.

IPsec ignora la modifica dei campi per la creazione dell'hash in un file

in modo sincronizzato (il mittente e il destinatario devono calcolare lo stesso hash sugli stessi parametri).

Modalità di trasporto IPsec

È utilizzato per la **sicurezza end-to-end**, cioè per gli host, non per i gateway (eccezione: traffico per il gateway stesso, ad esempio SNMP, ICMP). Il pacchetto originale viene tagliato in due parti e una



nuova intestazione viene inserita tra l'intestazione IPv4 e l'intestazione TCP/UDP. In questo modo, l'intestazione IPv4 assegnerà

che sta trasportando IPsec (invece di TCP/UDP) e poi, all'interno dell'intestazione IPsec, ci sarà un altro campo che dirà cosa viene effettivamente trasportato.

Figura 50 IPsec utilizzato per la sicurezza end-to-end in modalità trasporto

- **Pro:** è computazionalmente leggero
- **Contro:** nessuna protezione dei campi variabili dell'intestazione

Modalità tunnel IPsec

Viene utilizzato per **creare una VPN**, di solito tra **gateway**. Non viene creata tra router! Il termine corretto è gateway: è un punto di contatto tra una rete che si presume sicura e una rete non sicura. Il gateway aggiunge protezione creando il tunnel IPsec sicuro.

Il gateway prende il pacchetto originale con l'intestazione end-to-end e lo inserisce all'interno di un tunnel che ha come mittente questo gateway e come destinazione il

gateway che protegge la rete in cui si trova la destinazione. Con questo IP nel pacchetto IP, il gateway di invio applicherà IPsec.

- **Pro:** protezione completa del pacchetto, campi variabili dell'intestazione inclusi
- **Contro:** pesante dal punto di vista computazionale.

Sebbene non sia così comune, IPsec in modalità tunnel può essere utilizzato anche per la comunicazione end-to-end, anche se di solito viene adottato tra elementi di confine di reti più grandi: è anche chiamato **site-to-site VPN**, poiché le entità sono tipicamente intere reti.

AH

AH è l'acronimo di **Authentication Header**. Ci sono state tre versioni principali di IPsec.

Meccanismo della prima versione (RFC-1826):

- Integrità dei dati e autenticazione del mittente
- Supporto obbligatorio di keyed-MD5 (RFC-1828)
- Supporto opzionale di keyed-SHA-1 (RFC-1852)

Meccanismo della seconda versione (RFC-2402):

- Integrità dei dati, autenticazione del mittente e protezione (parziale) da attacchi replay
- HMAC-MD5-96
- HMAC-SHA-1-96

Il formato dell'intestazione che viene aggiunta al pacchetto IPsec ha:

- Il campo dell'**intestazione successiva** è uno pseudo-protocollo, quindi nell'intestazione IP sarà scritto che sta trasportando l'AH, ma poi all'interno dell'AH c'è il vero campo del pacchetto di trasporto;
- Parametro di **lunghezza** di 1 byte per descrivere la lunghezza del pacchetto;
- Byte **riservati** per usi futuri;
- **Indice dei parametri di sicurezza (SPI):** 32 bit per fare riferimento in modo rapido e semplice a tutti i parametri che è necessario verificare nel pacchetto;
- **Numeri di sequenza** per evitare attacchi di replay;
- **Integrity Check Value (ICV):** numero variabile di parole da 4 byte per fornire dati di

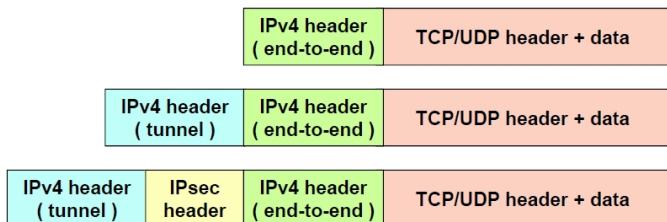


Figura 51 IPsec utilizzato in modalità tunnel per creare una VPN. IPsec non viene applicato direttamente al pacchetto end-to-end, ma a un tunnel tra i gateway.

Next Header	Length	reserved
Security Parameters Index (SPI)		
Sequence number		
.		
	<i>authentication data</i> (ICV, Integrity Check Value)	.
.		.

Figura 52 Formato dell'intestazione IPsec AH secondo RFC-4302

autenticazione (digest).

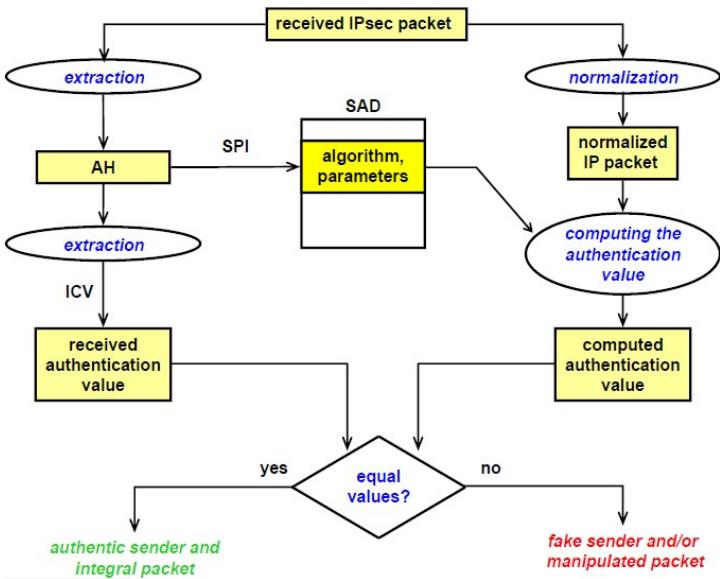


Figura 53 Elaborazione di un pacchetto IPsec ricevuto

Quando un pacchetto IPsec viene ricevuto, è protetto con AH. Si inizia con l'estrazione dell'AH e da esso si estrae l'**ICV**, che è il *valore di autenticazione ricevuto* (il digest calcolato dal mittente). Quindi, sul pacchetto ricevuto viene eseguita la **normalizzazione**, che significa mettere il pacchetto nelle stesse condizioni in cui si trovava al mittente per calcolare lo stesso tipo di hash. Una volta disponibile il pacchetto IP normalizzato, è necessario *calcolare il valore di autenticazione (ICV)*. A tale scopo, all'interno del *database dell'associazione di sicurezza (SAD)* viene utilizzato il *Security Parameter Index (SPI)*. Si tratta di un puntatore che indica l'algoritmo e i parametri da utilizzare. Questi parametri possono essere usati per calcolare il valore di autenticazione e poi si controlla se i due valori (quello calcolato e quello

ricevuti dal mittente) sono uguali. Se i due valori sono uguali, il *mittente è autentico e il pacchetto è integro*. Se i due valori non sono uguali, potrebbe esserci un *mittente falso e/o un pacchetto manipolato*.

Nell'immagine precedente viene specificato un mittente autentico, ma non c'è un punto in cui il mittente viene autenticato. Quindi, chi è il mittente autenticato in questo caso? La risposta sta nel fatto che viene utilizzata una voce specifica del DAU. Tale voce è stata negoziata con un nodo specifico. Per questo motivo, l'autenticazione è implicita nel processo. La vera autenticazione entra in gioco quando si crea l'associazione di sicurezza: è questo il momento in cui il mittente deve dimostrare la propria identità. L'associazione di sicurezza (SA) comporta questo tipo di autenticazione grazie all'utilizzo di algoritmi/parametri corretti.

Normalizzazione per AH

La normalizzazione viene eseguita sia al mittente che al destinatario perché il pacchetto deve essere nello stesso stato, altrimenti i due ICV sarebbero diversi. Ciò significa che:

- Il campo denominato *TTL* in IPv4 e il campo *Hop Limit* in IPv6 devono essere azzerati per eseguire il calcolo.
- Se il pacchetto contiene un'intestazione di routing nelle opzioni, allora:
 - Impostare il campo di destinazione sull'indirizzo della destinazione finale;
 - Imposta il contenuto dell'intestazione di instradamento sul valore che avrà a destinazione;
 - Impostare il campo Indice indirizzo al valore che avrà a destinazione.
- Azzeramento di tutte le opzioni con il bit C (*change en route*) impostato.

Il digest della chiave si basa su HMAC che ha un parametro additivo "96" alla fine. L'**HMAC-SHA1-96** viene generato nel modo seguente:

- Dato **M** normalizzarlo per generare **M'**
- Imbottire **M'** a un multiplo di 160 bit (aggiungendo 0x00 byte) per generare **M'p**
- Calcolo della base di autenticazione: **B = HMAC-SHA1 (K, M'p)** (dove K è la chiave concordata con il mittente)
- **ICV = 96 bit più a sinistra di B**

Poiché abbiamo detto che un digest lungo è necessario per evitare molti tipi di attacchi, la scelta di ottenere solo 96 bit sembra avventata: il motivo è che qui c'è un conflitto di interessi tra sicurezza e gestori di rete. I gestori di rete avrebbero bisogno di un header di **dimensioni fisse**, infatti, se si utilizzano algoritmi diversi

per il digest, questi producono digest di lunghezze diverse, il che rappresenta un problema per i produttori di router (che avrebbero un header di dimensioni variabili). Sarebbe quasi impossibile elaborare l'intestazione: ovviamente in questo modo si ha una minore resistenza all'attacco replay. Nella versione 3 di IPsec questo aspetto è stato migliorato.

ESP

Se si desidera la riservatezza, è necessario l'**Encapsulating Security Payload** (ESP). La prima versione (RFC-1827) prevedeva solo la riservatezza. Il meccanismo di base funziona con **DES-CBC** (RFC-1829), ma sono possibili anche altri meccanismi. La seconda versione prevedeva anche l'autenticazione, ma non per l'intestazione IP, quindi la copertura non è equivalente a quella di AH. Il vantaggio è che la **dimensione del pacchetto è ridotta e si risparmia un SA**.

RICORDA: IPsec è un'architettura che viene implementata con due tipi di pacchetti: AH o ESP, che possono essere utilizzati contemporaneamente.

L'ESP può essere utilizzato anche in modalità trasporto e in modalità tunnel.

ESP in modalità trasporto

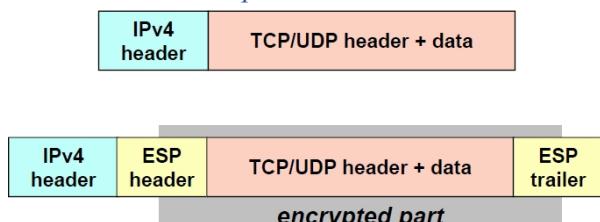


Figura 54 Utilizzo dell'intestazione ESP in modalità trasporto

ESP in modalità tunnel

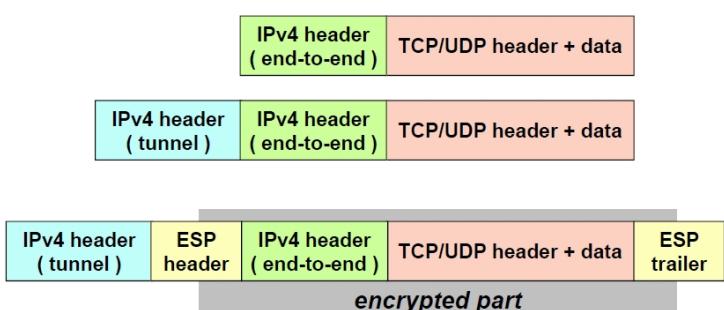


Figura 55 Utilizzo dell'intestazione ESP in modalità tunnel

ESP - Formato (II versione)

In esso sono in chiaro l'*indice dei parametri di sicurezza* (SPI) e il *numero di sequenza*. Tutto il resto sono dati criptati: è necessario avere la voce corrispondente all'SPI per leggere all'interno dei dati crittati.

Supponiamo di aver usato il **DES-CBC** e che il formato sia questo:

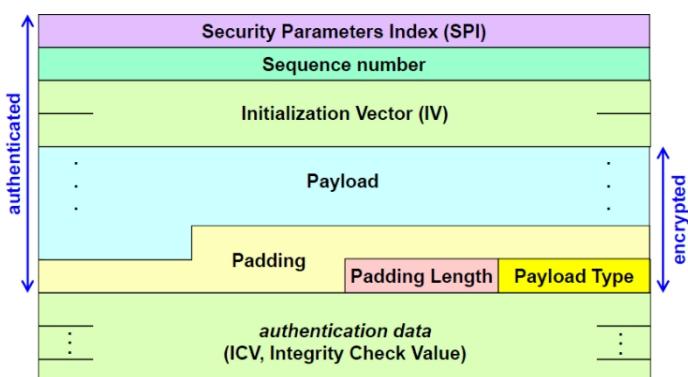


Figura 57 Formato dell'intestazione IPsec ESP-DES-CBC secondo RFC-4306

Se viene utilizzato in modalità tunnel, prima viene creato il tunnel, poi la protezione viene applicata al payload del tunnel. In questo caso, tutto il pacchetto originale viene crittografato, compresa l'intestazione end-to-end.

Security Parameters Index (SPI)
Sequence number
:
encrypted data
:

Figura 56 Formato dell'intestazione IPsec ESP secondo RFC- 2406

Poiché viene utilizzato il DES-CBC, è necessario un *vettore di inizializzazione (IV)*, che deve essere di 64 bit. Poi c'è il payload stesso. La parte che va dall'SPI al padding (per raggiungere un multiplo di 64 bit) è quella *autenticata*. Ci deve essere anche 1 byte per dichiarare la *lunghezza del padding* e 1 byte per il *Payload Type*, che è il protocollo di livello 4

che stiamo trasportando. Il fatto che il Payload Type sia criptato crea problemi ai gestori di rete, perché significa che se si utilizza un pacchetto IPsec che utilizza l'ESP, l'intermediario

I sistemi **non sono** in grado di vedere il protocollo di livello 4 e **non possono** eseguire QoS (o differenziazione del traffico), ad esempio per dare priorità a TCP o UDP. Se ESP viene utilizzato non solo per la riservatezza, ma anche per l'autenticità e l'integrità, alla fine c'è un numero variabile di parole di 4 byte contenenti l'ICV (Integrity Check Value).

Dettagli sull'implementazione di IPsec

Poiché gli algoritmi utilizzabili sono molti, la RFC-4308 definisce due suite di crittografia che chiunque dovrebbe implementare per l'interoperabilità:

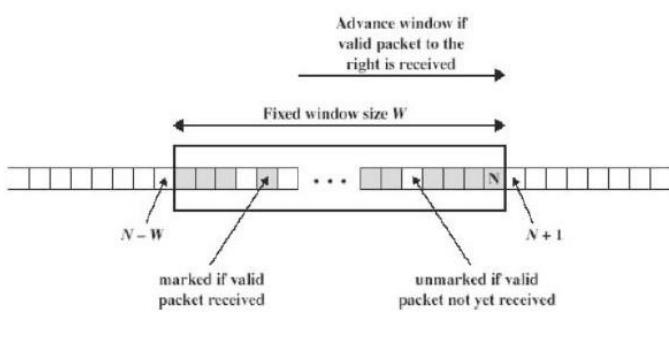
- **VPN-A** che utilizza ESP con 3DES-CBC e HMAC-SHA1-96: si tratta di una sorta di VPN legacy per la compatibilità con i vecchi sistemi.
- **VPN-B** che utilizza ESP con AES-128-CBC e AES-XCBC-MAC-96: è quello utilizzato attualmente.

È anche possibile utilizzare **algoritmi NULL** per ESP. È possibile specificare NULL per una delle due parti, autenticazione o privacy, ma **non contemporaneamente**. Questo permette di avere una sorta di compromesso tra "protezione e prestazioni". Per quanto riguarda il numero di sequenza, questo fornisce una protezione parziale contro gli attacchi di replay e funziona con una finestra minima di 32 pacchetti (64 consigliati).

Protezione IPsec replay (parziale)

Quando il mittente invia i pacchetti in sequenza, questi vengono numerati. Poi, dato che la rete utilizza il protocollo IP, i pacchetti possono essere persi, duplicati o arrivare fuori ordine, il che è un grosso problema. Se il destinatario ha ricevuto i pacchetti 0-1-2 e poi il 4, è possibile dichiarare che il pacchetto 3 è stato cancellato da un aggressore? Ovviamente non è possibile rispondere perché il pacchetto potrebbe essere perso. Inoltre, quando il ricevitore ha già ricevuto il pacchetto 53 e poi arriva il pacchetto 3, è possibile affermare che il pacchetto 3 è duplicato? No, perché potrebbe trattarsi di un pacchetto fuori servizio.

Per sapere se un pacchetto, che nella sequenza è precedente all'ultimo ricevuto, è un duplicato, l'unico modo è tenere un elenco dei pacchetti ricevuti e controllare se il pacchetto è già stato ricevuto o meno. Il ricevitore non può avere un elenco di tutti i pacchetti ricevuti, perché sarebbe un numero enorme.



Per questo motivo, si utilizza una finestra scorrevole. Esiste una **finestra fissa di dimensione W** in cui i pacchetti ricevuti sono contrassegnati, mentre i pacchetti bianchi sono quelli non ricevuti. Quando viene ricevuto un vecchio pacchetto, se è di uno slot che non è stato ricevuto viene accettato perché è un pacchetto fuori ordine. Se è già stato ricevuto, viene scartato perché è un duplicato (dalla rete o da un aggressore).

Se viene ricevuto un vecchio pacchetto (che non è all'interno della finestra), non c'è modo di controllarlo. È un grosso rischio perché se viene accettato potrebbe essere un attacco replay, mentre se viene scartato potrebbe esserci un problema di comunicazione.

Se il traffico che si sta proteggendo è TCP, accettare un pacchetto vecchio non è molto rischioso perché se viene accettato ma è già stato ricevuto al livello superiore, verrà scartato perché quel segmento sarà già stato elaborato. Al contrario, se si tratta di un pacchetto UDP **non dovrebbe mai essere accettato** e si dovrebbe chiedere al mittente di inviarlo di nuovo con un numero di sequenza più fresco.

La finestra procede, perché se tutti i pacchetti fino a 20 sono stati ricevuti e arriva il pacchetto 21, allora la finestra si sposta di un pacchetto sempre a destra. Ogni volta che viene ricevuto un nuovo pacchetto con un numero di sequenza maggiore di quello attuale, la finestra scorre. Non è possibile attendere i pacchetti non ancora ricevuti perché potrebbero essere persi (e IP non controlla la ritrasmissione).

IPsec v3

IPsec v3 cambia il paradigma circa l'uguale supporto alle due intestazioni, infatti ESP è obbligatorio e AH opzionale: questo significa che è possibile trovare implementazioni di IPsec che non supportano AH cioè integrità e authN solo per il payload, non per l'intestazione. IPsec v3 ha anche aggiunto il supporto per il multicast a sorgente singola. Inoltre, poiché IPsec è stato utilizzato principalmente per creare VPN site-to-site, ovvero canali con molto traffico, per evitare l'overflow dei numeri di sequenza IPsec v3 ha introdotto l'**ESN** (*Extended Sequence Number*), composto da 64 bit anche se all'interno dei pacchetti ci sono ancora solo 32 bit, perché i 32 meno significativi sono quelli che vengono trasmessi: questo è il default quando si utilizza **IKEv2**. Inoltre, invece di avere la crittografia separata del payload più il **MIC** (Message Integrity Code), è stato introdotto il supporto per la crittografia autenticata (**AEAD**) e alcuni chiarimenti su SA (Security Association) e SPI (Security Parameter Index) per ottenere una ricerca più veloce.

Gli algoritmi utilizzati in IPsec sono i seguenti:

- Per l'integrità e l'autenticazione:
 - (MAGGIO) *HMAC-MD5-96*;
 - (DEVE) *HMAC-SHA-1-96*;
 - (DOVREBBE+) *AES-XCBC-MAC-96*;
 - (DEVE) *NULL* (solo per ESP);
- Per la privacy:
 - (DEVE) *NULL*;
 - (OBBLIGATORIO - cioè sconsigliato) *TripleDES-CBC*;
 - (DOVREBBE+) *AES-128-CBC*;
 - (DOVREBBE) *AES-CTR*;
 - (NON DOVREBBE) *DES-CBC*.
- Per la crittografia autenticata (modalità AEAD):
 - *AES-CCM*;
 - *AES-CMAC*;
 - *ChaCha20 con Poly1305*.
- Per l'autenticazione e l'integrità, è possibile supportare digest più lunghi:
 - *HMAC-SHA-256-128*
 - *HMAC-SHA-384-192*
 - *HMAC-SHA-512-256*

TFC (Traffic Flow Confidentiality) in IPsec v3

Si tratta di un padding in ESP che viene inserito dopo il payload e prima del normale padding: è necessario per non rivelare la reale dimensione del payload nel pacchetto. Il destinatario deve essere in grado di calcolare la dimensione originale del payload (ad esempio, è possibile con i payload IP, UDP e ICMP grazie al campo "length"). Per le stesse ragioni di riservatezza, IPsec v3 introduce il supporto per i "**pacchetti fittizi**": si tratta di uno pseudo-protocollo annidato (header 59 successivo), necessario per poter continuare a trasmettere anche in assenza di dati reali da inviare. Ovviamente, ha senso utilizzare pacchetti fittizi solo se sono criptati. Il ragionamento alla base dei pacchetti fittizi è che a volte il solo fatto che i peer stiano comunicando può essere un'informazione preziosa.

Per esempio, immaginiamo che ci sia un generale che voglia impartire un comando di attacco e che ci siano varie squadriglie: se il nemico potesse vedere che il generale sta comunicando con un sito specifico, allora potrebbe supporre che in quel sito verrà eseguita un'azione. L'invio di pacchetti fittizi alle altre squadriglie renderà i messaggi indistinguibili per il nemico. Ovviamente, questo riduce le prestazioni globali aumentando i dati inviati sulla rete, ma è un prezzo da pagare per ottenere questo livello di riservatezza.

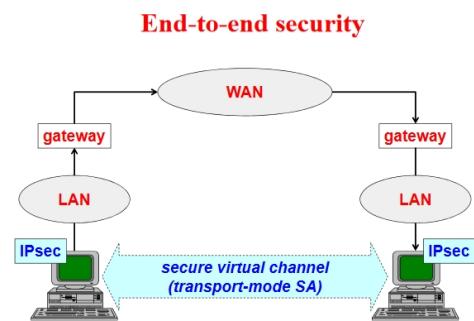
Modi per utilizzare IPsec

Sicurezza end-to-end

Consiste nell'attivare il modulo IPsec sui nodi finali che stanno comunicando. Questi nodi hanno bisogno di un canale virtuale sicuro, quindi creano una modalità di trasporto SA tra loro, in modo che i pacchetti siano protetti con il livello selezionato proprio quando lasciano l'interfaccia di rete del mittente. In questo modo, non ci si preoccupa che la LAN non sia sicura, che il gateway sia gestito da una parte non attendibile o che la WAN non sia attendibile.

Il grande vantaggio è che la sicurezza è implementata indipendentemente dal resto della rete: l'unico attacco possibile è il DoS. Il

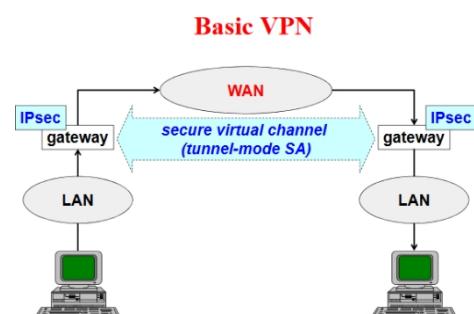
Lo svantaggio è che è necessario installare IPsec su entrambe le macchine che comunicano: oggi la maggior parte dei sistemi operativi supporta IPsec in modo nativo, ma su alcuni dispositivi il modulo non è presente in modo nativo, ad esempio: dispositivi mobili (Android e iOS) e sistemi embedded. Quindi, il problema è essenzialmente quello di considerare il tipo di dispositivi utilizzati, non solo per quanto riguarda la disponibilità del modulo software IPsec, ma anche per quanto riguarda la potenza di calcolo necessaria per utilizzarlo. Un altro aspetto critico riguarda la gestione della sicurezza: se IPsec viene utilizzato per proteggere tutti i computer di una grande LAN, è necessario un sistema efficiente per gestire le configurazioni IPsec su tutti i nodi. Infine, se per il canale virtuale sicuro si adotta ESP con algoritmo di crittografia non nullo, allora il traffico non può essere sniffato nemmeno dall'interno della LAN, per cui, ad esempio, l'adozione di un IDS (*Intrusion Detection System*) all'interno di una qualsiasi delle due LAN è impossibile, perché non si vedrebbe il reale contenuto di ciò che viene trasportato: in questo caso è necessario che l'IDS sia posizionato direttamente sui nodi, piuttosto che nella rete.



VPN di base

I moduli IPsec sono collocati direttamente sui gateway che proteggono la rete interna da quella esterna. In questo caso, poiché il canale deve proteggere tutto il traffico tra le due reti, deve essere un SA in modalità tunnel, perché i pacchetti provenienti da una rete all'altra devono essere incapsulati.

Il presupposto principale di questa architettura è che la rete interna sia sicura e affidabile, per cui l'unica preoccupazione deriva dagli attacchi sulla WAN. Ciò significa lasciare una porta aperta agli attacchi interni e non è più possibile effettuare l'autenticazione di i veri endpoint della comunicazione: solo gli elementi intermedi, che sono i gateway, sono autenticati. Questo modello è chiamato anche **VPN site-to-site**.



In questo schema, di solito i gateway non soffrono il problema di avere capacità IPsec, ma c'è comunque un problema di capacità computazionale: poiché i gateway devono eseguire il tunnelling per tutte le comunicazioni, potrebbero essere *sovraffaticati*. In genere, in questo caso i gateway sono dotati di potenti CPU o acceleratori hardware, come un HSM. D'altra parte, la gestione è notevolmente semplificata, perché non è necessario gestire le configurazioni per tutti i nodi finali, ma solo sui gateway. Poiché non esiste una sicurezza end-to-end tra i peer, esiste anche la possibilità di ispezionare il traffico interno.

Sicurezza end-to-end con VPN di base

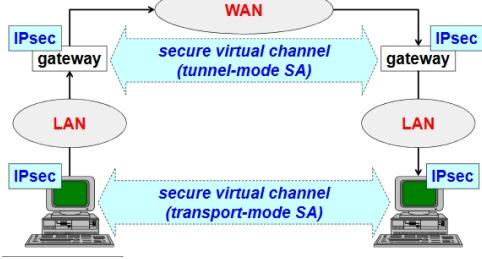
È un esempio del principio della "difesa in profondità", cioè della creazione di più linee di difesa. Qui c'è una doppia attivazione di IPsec: tra i nodi finali e tra i gateway. In questo modo è possibile avere due linee di difesa o anche bilanciare la sicurezza: ad esempio, la connessione end-to-end, cioè la modalità di trasporto SA, potrebbe essere utilizzata solo per l'autenticazione e l'integrità (ad esempio, per identificare chiaramente chi è il mittente), mentre la crittografia potrebbe essere attivata solo tra i gateway per proteggere dallo sniffing nella WAN e mantenere la possibilità di ispezionare il traffico sulla LAN.

Il problema è la gestione dell'intera struttura, cioè la gestione del numero massimo di sistemi (come nella prima architettura) e di tutti i gateway (come nella seconda).

Gateway sicuro

Il concetto di secure gateway è che ci sono utenti mobili, ad esempio dipendenti che viaggiano e hanno bisogno di connettersi alla rete aziendale interna. In questo caso, è possibile implementare IPsec sul dispositivo mobile dell'utente e creare un canale virtuale sicuro in modalità tunnel SA tra il dispositivo e il gateway aziendale. Questo permette di proteggere tutto il traffico dall'utente verso qualsiasi server interno alla rete aziendale; inoltre, in questo modo il gateway sarà anche in grado di eseguire l'autenticazione e l'autorizzazione.

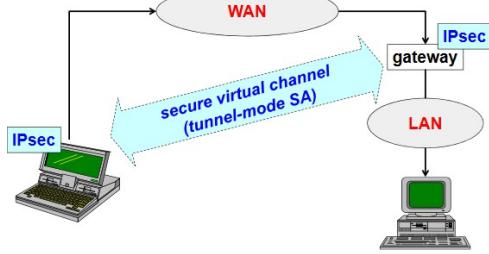
End-to-end security with basic VPN



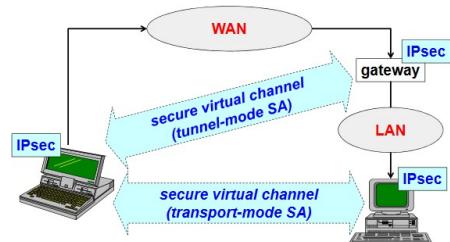
Accesso remoto sicuro

Anche in questo caso esiste la possibilità di una doppia linea di difesa: una modalità tunnel tra il nodo mobile e il gateway e un canale virtuale end-to-end tra il nodo mobile e la destinazione finale. In questo caso, tipicamente la modalità tunnel viene utilizzata solo per l'autenticazione e l'autorizzazione (per dare accesso alla rete interna), mentre la modalità di trasporto SA è tipicamente utilizzata per la protezione end-to-end, a seconda del tipo di protezione necessaria.

Secure gateway



Secure remote access



Gestione delle chiavi IPsec

È un componente importante che fornisce a tutte le parti le chiavi simmetriche utilizzate per l'autenticazione dei pacchetti ed eventualmente per la crittografia. Per distribuire queste chiavi, è possibile utilizzare un approccio OOB (out-of-band), ad esempio passando le chiavi **manualmente**. È fisicamente possibile se c'è un numero limitato di nodi e se c'è la possibilità di iniettare direttamente le chiavi a questi nodi. In presenza di molti nodi, è necessaria una distribuzione **automatica** delle chiavi **in banda**, che a sua volta richiede un protocollo.

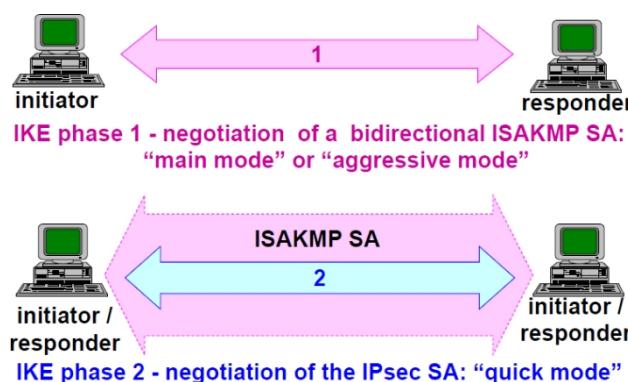
ISAKMP

Parte integrante dell'architettura IPsec è il protocollo **ISAKMP** (*Internet Security Association and Key Management Protocol*). Descritto in RFC-2408, contiene le *procedure necessarie per negoziare, impostare, modificare e cancellare un SA*. Purtroppo, non ci sono informazioni sulla chiave: il metodo di scambio della chiave non è fisso, e può essere un metodo OOB (e ISAKMP è usato solo per selezionare la chiave appropriata) o un metodo in-band usando **OAKLEY**, un protocollo per lo scambio autenticato di chiavi simmetriche.

IKE

La combinazione di *ISAKMP* e *OAKLEY*, ampiamente adottata, è stata ribattezzata **IKE** (*Internet Key Exchange*). È uno dei protocolli di sicurezza più complessi perché funziona nel modo seguente:

- Innanzitutto viene creato un SA per proteggere lo scambio ISAKMP;
- La SA creata viene utilizzata per proteggere la negoziazione della SA necessaria al traffico IPsec;
- Lo stesso ISAKMP SA può essere riutilizzato più volte per negoziare altri IPsec SA.



A sinistra è riportato un possibile schema delle operazioni IKE. Nella fase uno c'è un nodo chiamato **initiator** perché prende l'iniziativa di aprire il canale IPsec verso un'altra macchina chiamata **responder**. Nella fase 1 di IKE è possibile negoziare un SA ISAKMP bidirezionale per proteggere il traffico in entrambe le direzioni ed è possibile scegliere tra *modalità principale* e *modalità aggressiva*.

Una volta che la SA è stata creata, uno qualsiasi dei due nodi può agire come iniziatore per creare una SA IPsec. La fase due può essere negoziata utilizzando la *modalità rapida*

perché tutto il traffico è già protetto dal SA. Le principali modalità di funzionamento sono:

- **Modalità principale**
 - Richiede lo scambio di *6 messaggi* (→ piuttosto lento)
 - *Protegge l'identità delle parti*: Gli indirizzi IP sono ovviamente sempre visibili a un attaccante, ma ricordate che in IPsec l'autenticazione è data da ciò che è stato usato durante la fase di autenticazione per creare la chiave. Queste identità "crittografiche" non vengono rivelate.
- **Modalità aggressiva**
 - *3 messaggi (ma non protegge l'identità delle parti)*
- **Modalità rapida**
 - *3 messaggi*
 - *Negoziazione della sola SA IPsec*
- **Modalità nuovo gruppo**: utilizzata per comunicare all'altro peer una modifica dell'algoritmo o della chiave utilizzata per proteggere il traffico.
 - *Solo 2 messaggi*

Quando viene aperto un SA è necessaria l'autenticazione; esistono quattro modi per eseguire l'autenticazione:

- **Firma digitale**
 - *Non ripudio della negoziazione IKE, il che significa che non è possibile negare la richiesta di apertura del canale sicuro.*
- **Crittografia a chiave pubblica**
 - *Protezione dell'identità fornita in modalità aggressiva*
- **Crittografia a chiave pubblica riveduta**
 - *Meno costoso dal punto di vista computazionale, solo 2 operazioni a chiave pubblica*
- **Chiave precondivisa**
 - *L'ID della parte può essere solo il suo indirizzo IP (il che crea problemi con gli utenti mobili).*

Concentratore VPN

Oggi IPsec viene utilizzato soprattutto per creare VPN site-to-site. Ciò significa che esiste il problema delle prestazioni dei gateway. Per questo motivo, è stata proposta l'esistenza di un **concentratore VPN**, un dispositivo hardware speciale che funge da terminatore del tunnel IPsec utilizzato per l'accesso remoto di singoli client o per creare VPN site-to-site. Essendo implementato in hardware, ha prestazioni molto elevate a fronte di costi contenuti. Dovrebbe essere preso in considerazione in uno scenario aziendale/campus in cui viene scambiato molto traffico e molte persone svolgono il proprio lavoro da remoto.

Conclusioni di IPsec

IPsec può essere applicato **solo ai pacchetti unicast** (*no broadcast, no multicast, no anycast*) perché è necessario identificare il peer e scambiare una chiave; come abbiamo visto, IPsec v3 ha un'aggiunta per il multicast a sorgente singola, ma questa è un'eccezione. Inoltre, poiché è necessaria l'autenticazione reciproca, IPsec **si applica tra parti che hanno attivato una SA** tramite chiavi *condivise* o *certificati X.509*. In generale, IPsec è adatto a **gruppi "chiusi"**: IPsec non può essere utilizzato, ad esempio, per creare un servizio di e-commerce (perché gli utenti del servizio sono sconosciuti: non ci sono informazioni su chiavi o certificati).

Fondamentalmente, IPsec fornisce una certa sicurezza al traffico di livello superiore, che viene trasportato all'interno dei pacchetti IP. Tuttavia, molti protocolli trasportati su IP ereditano l'insicurezza di quest'ultimo, poiché gli **indirizzi IP non sono autenticati** e i **pacchetti non sono protetti** (se non si utilizza IPsec). Per questi motivi, tutti i protocolli che utilizzano l'IP come vettore possono essere attaccati. Normalmente, l'attaccante tende a colpire i protocolli che vengono trascurati, i cosiddetti protocolli "di servizio" (cioè quelli non applicativi, come ICMP, IGMP, DNS, RIP e così via).

"Protocolli di sicurezza del "servizio

Sicurezza ICMP

ICMP è il *protocollo di controllo e gestione di Internet*. È fondamentale per la gestione della rete e per questo motivo sono possibili molti attacchi perché **non ha autenticazione**. Le funzioni di ICMP sono le seguenti:

- *Richiesta/risposta ICMP echo*
 - Ping flooding o Ping bombing
- *Destinazione non raggiungibile (rete/host/protocollo/porta non raggiungibile)*
 - Poiché i pacchetti non sono autenticati, i nodi fasulli potrebbero inviare destinazione irraggiungibile al mittente, il quale chiuderà la comunicazione pensando che la destinazione sia davvero irraggiungibile: si tratta di un DoS.
- *Source quench (deprecato con RFC-6633, anno 2012):*
 - È stato pensato per fornire un meccanismo di controllo della congestione: i mittenti *devono* reagire ai messaggi ICMP Source Quench rallentando la trasmissione sulla connessione, perché i buffer intermedi sono pieni e i nodi intermedi non sono in grado di inviare i dati alla destinazione alla velocità richiesta.
 - Anche in questo caso, i messaggi ICMP Source Quench non hanno alcuna autenticazione, il che significa che potrebbero essere inviati pacchetti falsi con l'obiettivo di rallentare la destinazione (DoS).
- *Reindirizzamento*: è un messaggio ICMP che può essere inviato da un nodo intermedio quando rileva che il pacchetto ha preso una strada sbagliata.
 - È possibile creare un MITM logico: un attaccante può reindirizzare il mittente a un nodo maligno sotto il suo controllo e lì può eseguire l'attacco MITM;
- *Tempo superato per un datagramma*: si tratta di un messaggio ICMP normalmente inviato da un nodo intermedio quando sta elaborando un pacchetto con TTL=0
 - La ricezione di questo messaggio di solito indica la presenza di loop nel piano di instradamento, quindi l'host chiude la connessione.
 - La falsificazione di questo messaggio può causare DoS per il mittente.

Non è possibile proteggere ICMP con IPsec e questo tipo di problemi è sempre possibile. L'unica cosa possibile è collaborare con il gestore della rete e tenere sotto controllo tutti i pacchetti ICMP che attraversano la rete.



Attacco dei puffi

ICMP può essere utilizzato anche per eseguire l'attacco *Smurf* (un tipo di attacco DoS). Esiste un bersaglio (A) che viene attaccato creando una richiesta di eco ICMP (Ping) con un mittente falso, come se il ping fosse inviato dalla vittima. La destinazione è l'indirizzo broadcast di un'intera rete. Viene inviato un ping a tutti i nodi all'interno di quella rete, che prende il nome di **riflettore**, perché invia una risposta *pong* (*ICMP echo reply*) per questo ping. Poiché è indirizzato all'indirizzo di broadcast, tutti i nodi attivi della rete risponderanno. Con un singolo pacchetto ping,

è possibile raggiungere 60k risposte. Il target finale sarà sommerso di messaggi, mentre la rete del riflettore sarà piuttosto occupata perché ci sarà molto traffico ICMP da inoltrare.

Per prevenire questo attacco, in genere per gli attacchi esterni la rete **rifiuta i pacchetti IP broadcast al confine**. Immaginiamo un router di confine con interfaccia *serial0* nella rete esterna. È possibile dichiarare:

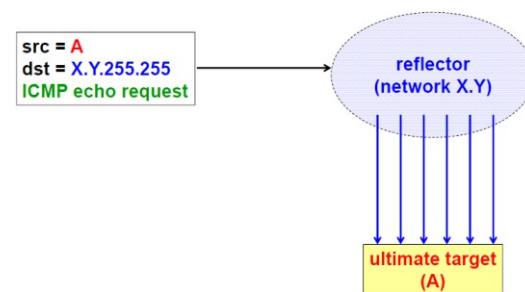
```
interfaccia serial0  
no ip directed-broadcast
```

In questo modo tutti i pacchetti IP trasmessi verranno scartati. Ma questa è una soluzione per gli attacchi esterni. La rete è ancora sensibile a un attacco proveniente dall'interno della intranet (la trasmissione in LAN non può essere disabilitata perché è richiesta dai protocolli). In quest'ultimo caso, l'unico modo per contrastare una fonte interna di smurfing è identificare l'attaccante tramite gli strumenti di gestione della rete e quindi cercare di bloccare la sua macchina.

Attacco Fragle

Fraggle è un attacco simile allo smurfing, ma viene eseguito utilizzando UDP, anziché ICMP. La filosofia è la stessa: un nodo falso invia una *richiesta di echo UDP*.

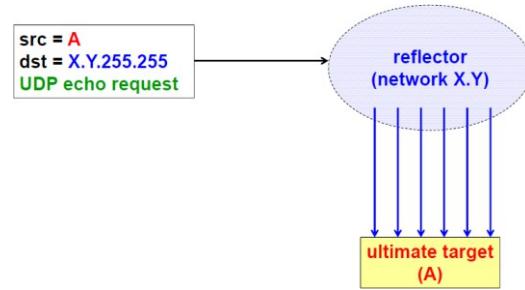
Oggi queste richieste sono disabilitate per impostazione predefinita. Pertanto, il successo di questo tipo di attacco dipende dal numero di client all'interno della rete del riflettore che hanno la *richiesta di echo UDP* abilitata.



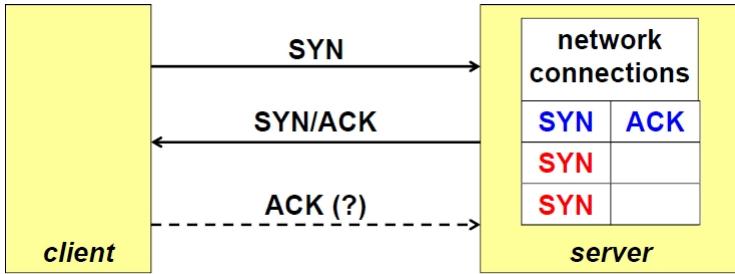
Avvelenamento ARP

ARP è il protocollo di risoluzione degli indirizzi utilizzato per scoprire l'indirizzo di livello 2 sfruttando la conoscenza degli indirizzi di livello 3. Il risultato viene memorizzato nella tabella ARP. Il risultato viene poi memorizzato nella tabella ARP. L'**avvelenamento ARP** viene eseguito in questo modo:

- I nodi accettano la risposta ARP anche se non hanno inviato una richiesta ARP, perché potrebbero evitare di dover inviare una richiesta ARP in futuro per contattare quel nodo;
- È quindi possibile inviare ai nodi risposte ARP errate non richieste;
- I nodi sovrascrivono le voci ARP statiche con quelle dinamiche (ottenute dalle risposte ARP).
 - *La maggior parte degli stack non controlla che l'indirizzo HW di origine all'interno del campo ARP coincida con il campo di origine del pacchetto 802.3.*
- Viene utilizzato da molti strumenti di attacco (ad esempio, Ettercap).



Inondazione TCP SYN



Ricordando l'handshake a tre vie per aprire una connessione TCP, viene inviato un primo pacchetto contenente il flag *SYN*. Il destinatario risponde con un segmento contenente entrambi i flag *ACK* e *SYN*. Normalmente il client chiude/apre il canale con un *ACK* finale.

Il client può essere malintenzionato e inviare un file

SYN ma non invia mai l'*ACK* finale. Il

Il server passa a un'altra riga della sua tabella delle connessioni e il client torna al primo passo e invia nuovamente un altro *SYN*. A un certo punto la tabella delle connessioni sarà piena (dimensione massima raggiunta) e non sarà in grado di aprire nuove connessioni per gli utenti reali. Questo sarà un DoS per i client che desiderano connettersi al server.

Si noti che l'*ACK* finale mancante può essere dovuto anche a un problema di rete. Nei documenti TCP-IP è scritto che dopo un certo numero di secondi (in genere 75s) il server controlla la sua tabella delle connessioni e, se sono state aperte mezze connessioni, queste vengono azzerate.

Per proteggersi dal SYN flooding è possibile:

- **Diminuire il timeout**
 - C'è il rischio di cancellare le richieste di clienti validi ma lenti.
- **Aumentare le dimensioni della tabella**
 - Può essere aggirato inviando più richieste
- **Utilizzare un router, davanti al server, come "intercettatore SYN".**
 - Sostituisce il server nella prima fase
 - Se l'handshake si conclude con successo, trasferisce il canale al server.
 - Timeout "aggressivo" (rischioso!)
- **Utilizzare un router come "monitor SYN"**
 - Uccide le richieste di connessione in attesa (RST).

Se il gestore della rete nota un'enorme quantità di *SYN* (più del solito), è il momento giusto per iniziare a indagare.

Cookie SYN

Oggi questa è la soluzione migliore contro il SYN flooding. Si tratta di un'idea di D.J. Bernstein, il quale ha notato che l'attacco è possibile perché, quando il server riceve un *SYN*, sta memorizzando qualcosa in una tabella su quel client. L'idea non è quella di mantenere lo stato della connessione all'interno del server, ma all'interno del client. L'idea utilizza il numero di sequenza TCP del pacchetto *SYN-ACK* per trasmettere un cookie (keyed-digest) al client e riconoscere successivamente i client che hanno già inviato il *SYN*, senza memorizzare alcuna informazione su di loro sul server. Questo è possibile perché un client risponde con un *ACK* che ha il numero di sequenza + 1 (che significa il keyed-digest + 1) e il server controllerà se il keyed-digest è uno valido fatto in precedenza da lui stesso. Più in dettaglio, il numero di sequenza iniziale del server è costruito come segue:

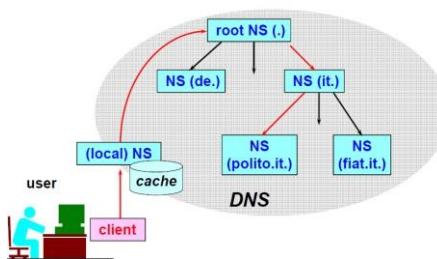
- sia t un timestamp che si incrementa lentamente; allora $t \bmod 32$ fornisce i primi 5 bit del numero di sequenza;
- Sia m il valore della dimensione massima del segmento (MSS) che il server avrebbe memorizzato nella voce della coda *SYN*;
- Sia s il risultato di una funzione hash crittografica calcolata sull'indirizzo IP e sul numero di porta del server, sull'indirizzo IP e sul numero di porta del client e sul valore t . Il valore restituito s deve essere un valore a 24 bit.

Il server, dopo aver ricevuto l'ACK e aver recuperato il digest con chiave:

- Controlla il valore **t** rispetto all'ora corrente per verificare se la connessione è scaduta.

- Ricompon s per determinare se si tratta effettivamente di un cookie SYN valido.
- Decodifica il valore m dalla codifica a 3 bit del cookie SYN, che può quindi utilizzare per ricostruire la voce della coda SYN.

Sicurezza DNS



Il **DNS** è il *sistema dei nomi di dominio*, che fornisce la traduzione dei nomi in indirizzi, ma svolge anche il lavoro inverso. È un servizio fondamentale perché ogni utente utilizza i nomi al posto degli indirizzi IP per accedere ai siti web. Funziona tramite **interrogazioni**, eseguite sulla porta *53/UDP*, e **trasferimenti di zone** (trasferimenti di informazioni tra server) sulla porta *53/TCP*. Di per sé, **non ha alcuna sicurezza intrinseca**: DNS-SEC è in fase di sviluppo e non è ancora stato implementato.

L'architettura del DNS è quella rappresentata nell'immagine qui sopra. Si tratta di una disposizione gerarchica di server, a partire da quello locale (cioè all'interno della rete del client). Se l'utente vuole accedere a un sito web, il DNS locale controlla la sua cache. In caso di *mancanza di cache*, il DNS locale interroga il *NS radice*, che in genere non ha la risposta, perché la radice gestisce solo domini di primo livello noti come punti ("."). La domanda verrà quindi reinviata dal NS radice al server di dominio giusto (prima a *it.*, poi, ad esempio, a *polit.it.*) per ottenere la risposta.

L'immagine sopra raffigura il percorso logico da seguire per recuperare l'indirizzo IP, ma in realtà il server dei nomi risponde sempre al DNS locale, che in genere salva nella cache le nuove query risposte.

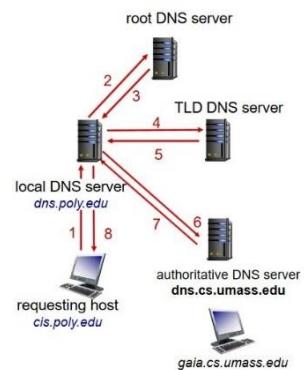
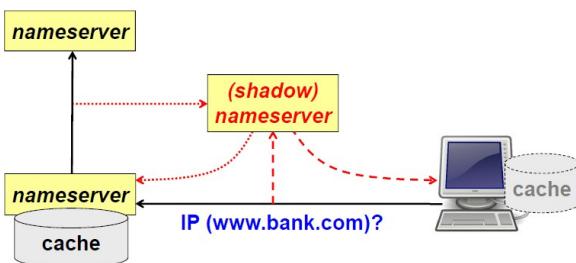


Figura 58 Risoluzione dei nomi DNS con query iterata

Server ombra DNS



Il primo tipo di attacco è un server DNS ombra/falso. Se qualcuno riesce a intercettare e a sniffare la richiesta al DNS locale, può eventualmente fornire anche una risposta falsa, in modo da fornire un indirizzo IP falso alla vittima.

Al contrario, se qualcuno riesce a intercettare la richiesta che va dal DNS locale verso il Root DNS, è possibile fornire una risposta falsa al DNS locale,

che fornirà il falso indirizzo IP a tutti i client della rete che forniranno la query DNS.

Microsoft ha implementato in Windows una cache per il DNS, che viola le regole generali del DNS. In questo caso, un attacco di questo tipo è molto più efficace perché la cache sul client non viene aggiornata spesso.

Avvelenamento della cache DNS



Il DNS ha una cache per evitare di ripetere le domande. In questo tipo di attacco viene creato un dominio (www.lioy.net). Ogni volta che viene creato un dominio è necessario anche un *nameserver*. In questo caso si tratta di un nameserver pirata, perché ogni volta che qualcuno chiede il nameserver (ad esempio, l'indirizzo di www.lioy.net) il nameserver pirata risponde con quello corretto, inserendo anche altre informazioni sbagliate. Se il nameserver vittima non è programmato e configurato correttamente, accetterà le risposte aggiuntive e

Figura 59 Avvelenamento della cache DNS attraverso la creazione di un nameserver (pirata) e la fornitura di risposte errate

sovrascriverà (se già memorizzate) le informazioni disponibili nella propria cache.

Il problema di questo attacco è che la vittima deve richiedere la risoluzione di un indirizzo pirata specifico. Per questo motivo, è possibile eseguire un'altra versione del cache poisoning.

Avvelenamento della cache DNS (versione 2nd)

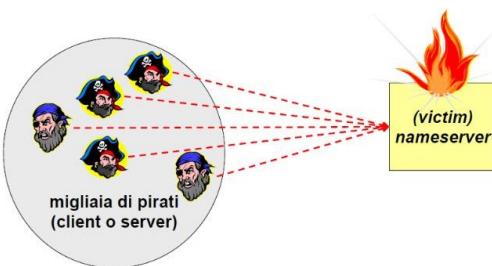
In questa versione l'aggressore dispone di un client DNS pirata e chiede al server dei nomi ricorsivo della vittima la risoluzione dell'indirizzo. Poiché la ricerca richiederebbe tempo, l'aggressore invia la risposta con un indirizzo sorgente falso, come se l'indirizzo provenisse dal NS autoritario.

Questo attacco è molto più efficace in quanto non è necessario che la vittima venga ingannata per richiedere la risoluzione del nameserver del pirata.



Figura 60 Avvelenamento della cache DNS mediante l'impostazione di un client DNS e la fornitura di domande e risposte errate con spoofing dell'IP

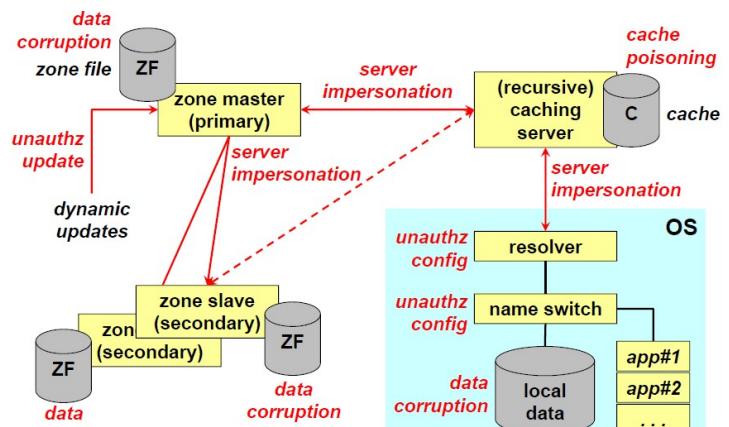
(DNS) folla di flash



L'unico modo possibile per un attacco DoS al DNS è avere molti client che pongono domande a un server dei nomi. Il DNS è l'obiettivo di molti attacchi, perché se il server dei nomi viene bloccato, tutti i domini collegati a quel server diventano irraggiungibili.

Traduzione nome-indirizzo

Naturalmente, la richiesta di traduzione nome-indirizzo parte da un dispositivo che esegue un sistema operativo. All'interno del sistema operativo varie applicazioni devono risolvere i nomi ottenendo l'indirizzo corrispondente. Per farlo, il sistema operativo sfrutta un componente chiamato **name switch**, in grado di decidere quale servizio viene utilizzato per passare da un nome all'indirizzo corrispondente. Ad esempio, il file *etc/hosts*, che è un dato locale. Se qualcuno è in grado di aggiungere una coppia nome-indirizzo all'interno di tale file, è possibile forzare le applicazioni a connettersi a un server fasullo.



Il name switch decide tipicamente la priorità tra il file interno e un servizio esterno. Il servizio esterno più comune è quello a cui si rivolge il resolver, cioè un componente che implementa il client DNS e può dialogare con l'infrastruttura DNS.

Se il nome non viene trovato nei dati locali, il resolver si connette a un servizio esterno (tipicamente un server di caching). Il server di caching può essere eventualmente ricorsivo, il che significa che si occupa da solo di porre tutte le domande. Il server di caching contatterà molti server, secondo la gerarchia descritta in precedenza, che si collegano a una zona specifica. Ogni zona è gestita da un master (*server dei nomi*)

primario) e da molti slave (*server dei nomi secondario*) che mantengono un database chiamato *file di zona*, che contiene la corrispondenza tra nomi e indirizzi. Il server di caching può contattare il server primario o i server secondari (nel caso in cui il primario non sia disponibile o sia sovraccarico).

Normalmente, il master di zona viene aggiornato manualmente dall'amministratore. Purtroppo, Microsoft ha aggiunto la possibilità di eseguire aggiornamenti dinamici direttamente dai client. Quando si aderisce a una rete Microsoft e viene assegnato un indirizzo, è possibile informare il master DNS dell'indirizzo IP ricevuto: questo è piuttosto pericoloso perché consente una modifica incontrollata dei dati DNS.

I possibili attacchi a questa infrastruttura sono:

- **Corruzione dei dati locali**: se un aggressore ottiene l'accesso (ad esempio, attraverso una tastiera o un malware) a un client, può cercare di inserire tra i suoi dati locali alcune informazioni per produrre una mappatura errata;
- **Configurazione non autorizzata** del name switch/resolver (invece di puntare al server dei nomi esterno effettivo, è possibile puntare a uno falso che fornirà risposte errate);
- Nella comunicazione tra il resolver e il server e tra il server di caching e il server di zona può verificarsi l'**impersonificazione del server** (cioè qualcuno che si spaccia per il vero server), poiché il DNS non prevede alcun tipo di autenticazione;
- Ogni volta che c'è una cache, potrebbe esserci anche un **avvelenamento della cache**;
- **La corruzione dei dati** può avvenire anche nel file di zona se l'attaccante riesce a penetrare il master di zona;
- **Gli aggiornamenti** dinamici potrebbero essere eseguiti da nodi **non autorizzati**;
- Poiché il master invia ai server secondari una copia del file di zona, anche in questo caso potrebbe verificarsi un'**impersonificazione del server**, il che significa che un falso master potrebbe inviare ai server secondari una copia errata.

Come si è visto, ci sono molti attacchi locali (come la corruzione dei dati) e alcuni sono attacchi di rete dovuti essenzialmente alla mancanza di autenticazione.

Il DNSsec è necessario

Nel febbraio 2008 Dan Kaminsky, un ricercatore, ha scoperto un nuovo attacco che rende l'avvelenamento della cache più semplice da eseguire, più difficile da evitare e applicabile anche ai record NS di livello 1st (ad esempio, .com, .it). Nel luglio 2008 sono usciti i primi avvisi e le prime patch e c'è stato un intervento di Kaminsky al Black Hat. Nel settembre 2008 gli Stati Uniti hanno reso obbligatorio l'uso di DNSsec per il dominio .gov a partire da gennaio 2009.

Oggi non esiste ancora una protezione sui domini .it (come in molti altri Paesi del mondo).

DNSsec

DNSsec è fondamentalmente la firma digitale dei record DNS che rende impossibile la creazione di risposte false. Anche se un record è firmato digitalmente, c'è un altro problema: *chi è "autorevole" per un certo dominio?* Ad esempio, se si riceve un voto dal Prof. Lioy, allora è legittimo, perché egli ha il diritto di firmare i voti, ma può apporre la firma digitale solo sui voti dei corsi di Lioy. La firma digitale non è sufficiente, è necessaria un'infrastruttura per stabilire chi è autorevole e chi ha il diritto di firmare digitalmente una risposta. Ogni firma digitale richiede anche un certificato: *qual è la PKI* (certificati, root CA fidata)?

Il problema è ancora più grande perché molti server in tutto il mondo hanno una gestione complessa dell'infrastruttura DNS. Per questo motivo, è necessario:

- *Firme gerarchiche e delegate*
- *Firme distribuite*

Un altro problema importante riguarda la gestione dei nomi inesistenti:

- *Anche l'ASSENZA di un record deve essere firmata, il che è difficile.*
- *Ciò richiede l'ordinamento dei record*

Alcuni problemi con DNSsec

- **La firma è disponibile solo sulle risposte e non sulle domande**, quindi non è possibile sapere chi ha posto la domanda;
- **Non esiste una CA radice**, ma le chiavi di livello 1 sono distribuite OOB, il che rende i nodi possibili bersagli di un attacco se le chiavi possono essere modificate;

- **Non c'è sicurezza nel dialogo tra il client DNS e il server DNS (locale).**
 - Utilizzare IPsec, TSIG o SIG(0), che sono stati proposti ma non implementati;
- **La firma che deve essere eseguita dal server DNS porta a:**
 - Overhead computazionale;
 - Overhead di gestione (è necessario un host crittografico sicuro on-line)
- **Dimensioni maggiori del record**, dovute alla presenza della firma e al fatto che è necessaria anche per allegare il certificato;
- **Scarsi risultati sperimentali** su quale sia la configurazione corretta e quale sia l'effettiva perdita di prestazioni;

Sicurezza dell'instradamento

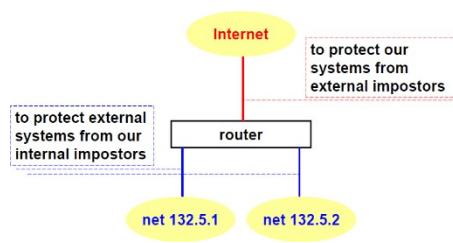
Normalmente, **l'accesso ai router** per la gestione del sistema è poco sicuro, ad esempio tramite *telnet* o *SNMP (Simple Network Management Protocol)*. Oltre alla gestione dei router, vi è una bassa sicurezza nello **scambio di tabelle di routing**:

- Autenticazione basata su indirizzi IP. Questo può essere sfruttato da un aggressore per inserire tabelle di routing false;
- La protezione opzionale con un digest con chiave non viene in genere implementata perché:
 - È necessaria una chiave condivisa! Tuttavia, poiché è condivisa, se viene distribuita a troppi router, questi possono impersonarsi a vicenda;
 - La gestione delle chiavi è necessaria! Poiché la chiave deve essere distribuita a molti router OOB. Infine, l'attacco di reindirizzamento su ICMP consentirà di variare l'instradamento dinamico sui nodi finali.

Protezione da spoofing IP

Non è possibile impedire a un host di cambiare il proprio indirizzo, ma è possibile cercare di limitare la diffusione dell'IP spoofing. Esistono alcune RFC che suggeriscono ai gestori di rete come aumentare la protezione dallo spoofing IP, sia per proteggere un host da impostori esterni sia per proteggere il mondo esterno da impostori interni alla rete locale.

- RFC-2827 "Filtraggio in ingresso alla rete: sconfiggere gli attacchi Denial of Service che utilizzano lo spoofing dell'indirizzo sorgente IP".
- RFC-3704 "Filtraggio in ingresso per le reti multihomed".
- RFC-3013 "Servizi e procedure di sicurezza raccomandati dai fornitori di servizi Internet".



Osservando l'immagine, è possibile notare due reti: *132.5.1* e *132.5.2* (sono due sottoreti collegate a un router di confine connesso a Internet). Sulla linea rossa, che collega il router a Internet, si vuole proteggere il sistema da impostori esterni. In genere, da Internet è possibile ricevere qualsiasi indirizzo, con una notevole eccezione: non è possibile ricevere alcun indirizzo che appartenga alle reti interne (non *132.5.1* e *132.5.2*).

indirizzi). Sulle connessioni tra router e sottoreti, invece, è possibile ricevere solo i pacchetti che appartengono alla rete specifica.

Inserendo dei filtri sulle linee, è possibile proteggersi da impostori sia esterni che interni. Se tutti i provider applicassero questa semplice regola, sarebbe possibile evitare lo spoofing IP a livello globale.

La possibile sintassi (linguaggio CISCO) in cui è possibile applicare i filtri è quella riportata nella figura a destra.

```

access-list 101 deny ip
 132.5.0.0 0.0.255.255 0.0.0.0 255.255.255.255
interface serial 0
ip access-group 101 in

access-list 102 permit ip
 132.5.1.0 0.0.0.255 0.0.0.0 255.255.255.255
interface ethernet 0
ip access-group 102 in

access-list 103 permit ip
 132.5.2.0 0.0.0.255 0.0.0.0 255.255.255.255
interface ethernet 1
ip access-group 103 in

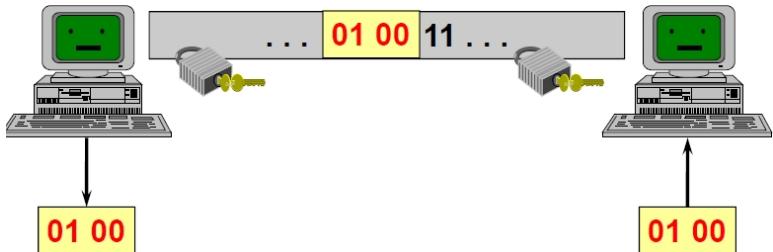
```

Sicurezza delle applicazioni di rete

La situazione standard in assenza di misure è molto negativa, perché la maggior parte dei sistemi utilizza un'autenticazione molto debole basata su nome utente e password (la password può essere intercettata) o sull'indirizzo IP (spoofing dell'IP). Anche se si utilizzano sistemi più forti (ad esempio, OTP, challenge response), anche se questi prevengono i problemi di autenticazione, esistono problemi con i dati: *snooping/forging dei dati, shadow server/MITM, replay, filtraggio*.

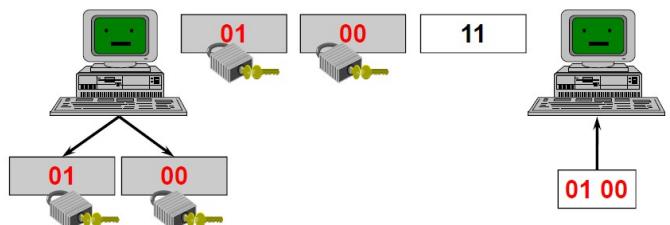
Ci sono due possibili approcci diversi per evitare i problemi:

- **sicurezza del canale:** significa che due nodi, prima di iniziare la comunicazione, negoziano algoritmi, parametri e chiavi per proteggere l'intero traffico che



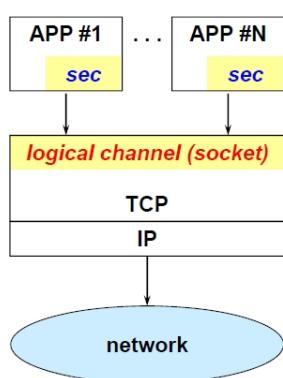
saranno inviati tra loro attraverso un canale sicuro. Poiché tutte queste caratteristiche vengono negoziate prima di trasmettere i dati, è possibile ottenere autenticazione **singola o reciproca, integrità e privacy**. Il problema principale è che queste caratteristiche sono fornite **solo durante il transito all'interno del canale di comunicazione**. Quando i dati escono dal canale sicuro, non sono più protetti e non c'è possibilità di ottenere *il non ripudio*. Poiché si tratta di una soluzione molto semplice da implementare, perché può essere realizzata anche a livello di sistema operativo, è una soluzione molto adottata perché non richiede quasi nessuna modifica alle applicazioni. Guardando l'immagine ci sono anche altri bit (i due 1 in nero non fanno parte del messaggio dell'utente) trasmessi sul canale e, per questo motivo, ottengono la sicurezza anche se non ne hanno bisogno;

- **sicurezza dei messaggi/dati:** ogni dato viene protetto individualmente avvolgendolo in un contenitore sicuro (ad esempio, PK-SSL). In questo caso è il mittente a creare la protezione e i dati che non richiedono sicurezza non sono protetti. In questo caso si ottiene solo un'autenticazione **singola**,



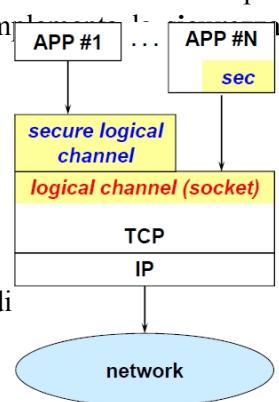
ma hanno ancora **l'integrità e la privacy contenute nel messaggio**. La protezione rimane anche quando i dati escono dalla rete e vengono memorizzati a destinazione: questo permette il **non ripudio**, ma richiede alcune modifiche alle applicazioni.

È possibile combinare i due approcci e avere dati sicuri all'interno di un canale sicuro: tuttavia, l'utilizzo di un canale di rete sicuro è la scelta preferibile, soprattutto per le applicazioni in rete come il Web.



Ogni applicazione di rete ha una parte comune che è fondamentalmente lo stack TCP/IP con l'interfaccia logica denominata **socket** (che consente di inviare e ricevere dati TCP o UDP in modo standard). In questo modo, quando le applicazioni hanno bisogno di sicurezza, hanno qualcosa al loro interno per implementare la parte di sicurezza. Ogni applicazione impiega la sicurezza **internamente** e la parte comune è limitato ai canali di comunicazione (ad esempio, il socket). Potrebbero esserci **errori di implementazione** (inventare protocolli di sicurezza non è semplice) e **l'interoperabilità non è garantita** a causa di possibili interpretazioni diverse delle specifiche.

Questa era una soluzione ma oggi non è più largamente utilizzata, perché per motivi di compatibilità è implicito acquistare tutto dallo stesso fornitore.



L'altro modo consiste nell'implementare la **sicurezza esternamente** alle applicazioni. Il **livello di sessione** sarebbe il luogo ideale da utilizzare per implementare molte funzioni di sicurezza, ma la sessione

non esiste nel TCP/IP. Poi, qualcuno ha proposto il **livello "canale logico sicuro"**. Si tratta di qualcosa che utilizza i socket per inviare dati arricchiti da una certa protezione. Semplifica il lavoro degli sviluppatori di applicazioni, evita errori di implementazione e spetta all'applicazione decidere se selezionarlo o meno (nessun problema di compatibilità). Oggi è uno standard de facto.

Protocolli di canale sicuro

- **SSL/TLS**
 - Il più utilizzato: inizialmente era SSL, poi ribattezzato TLS.
- **SSH**
 - È stato un prodotto di successo (soprattutto nel periodo in cui l'esportazione di prodotti crittografici statunitensi era limitata), ma oggi è un prodotto di nicchia da utilizzare solo in alcuni ambienti ristretti.
- **PCT**
 - Proposto da Microsoft come alternativa a SSL. È uno dei pochi fallimenti di Microsoft perché non è mai diventato famoso e MS ha abbandonato il PCT dai suoi prodotti.

SSL (Secure Socket Layer)

Inizialmente chiamato **Secure Socket Layer**, SSL è stato proposto da *Netscape Communications* (inventore del primo browser grafico per computer) che aveva bisogno di un protocollo sicuro per implementare il commercio elettronico sul web. SSL è un **canale di trasporto sicuro** (a livello di sessione) con alcune proprietà:

- **Autenticazione tra pari (server, server e client)**
 - L'autenticazione del server è obbligatoria all'apertura del canale SSL, a causa della sua origine: era necessaria per il commercio elettronico. Opzionalmente è possibile richiedere anche l'autenticazione del client. Ricordate che è a livello di sessione, il che significa che non sono richiesti nome utente/password.
- **Riservatezza dei messaggi**
 - La riservatezza è facoltativa perché forse l'autenticazione è sufficiente.
- **Autenticazione e integrità dei messaggi**
 - L'autenticazione e l'integrità sono obbligatorie perché servono a dimostrare chi ha creato il messaggio e che il messaggio non è stato modificato durante la trasmissione.
- **Protezione contro gli attacchi replay e di filtraggio**
 - Se qualcuno prende una copia del messaggio e la invia nuovamente al canale, verrà rilevato come un attacco replay. Allo stesso modo, se un nodo intermedio cancella (filtra) una parte dei dati, il ricevitore rileverà che manca una parte.

Dal momento che queste caratteristiche sono state poste sopra il TCP, il successo di SSL è dovuto al fatto che è facilmente applicabile a tutti i protocolli basati su TCP: *HTTP*, *SMTP*, *NNTP*, *FTP*, *TELNET*. Un esempio è il famoso *HTTPS* su *443/TCP*.

nsiops	261/tcp # IIOP Name Service over TLS/SSL
https	443/tcp # http protocol over TLS/SSL
smt�	465/tcp # smtp protocol over TLS/SSL (was ssmt�)
nntps	563/tcp # nntp protocol over TLS/SSL (was snnntp)
imap4-ssl	585/tcp # IMAP4+SSL (use 993 instead)
sshell	614/tcp # SSLshell
ldaps	636/tcp # ldap protocol over TLS/SSL (was sldap)
ftps-data	989/tcp # ftp protocol, data, over TLS/SSL
ftps	990/tcp # ftp protocol, control, over TLS/SSL
telnets	992/tcp # telnet protocol over TLS/SSL
imaps	993/tcp # imap4 protocol over TLS/SSL
ircs	994/tcp # irc protocol over TLS/SSL
pop3s	995/tcp # pop3 protocol over TLS/SSL (was spop3)
msft-gc-ssl	3269/tcp # MS Global Catalog with LDAP/SSL

Figura 61 Porte ufficiali per applicazioni SSL

SSL - autenticazione e integrità

L'autenticazione tra pari al momento dell'impostazione del canale viene eseguita con un'autenticazione forte:

- Il server si autentica inviando la propria chiave pubblica (certificato X.509) e rispondendo a una sfida asimmetrica implicita.
- L'autenticazione del cliente (con chiave pubblica, certificato X.509 e sfida esplicita per far sì che il cliente utilizzi la chiave privata al fine di dimostrarne il possesso) è opzionale (poiché molte persone non possiedono un certificato X.509).

Se l'autenticazione del peer fallisce, il canale non viene aperto. Se invece ha successo, tutti i dati sono protetti. Per l'autenticazione e l'integrità dei dati scambiati sul canale, il protocollo utilizza:

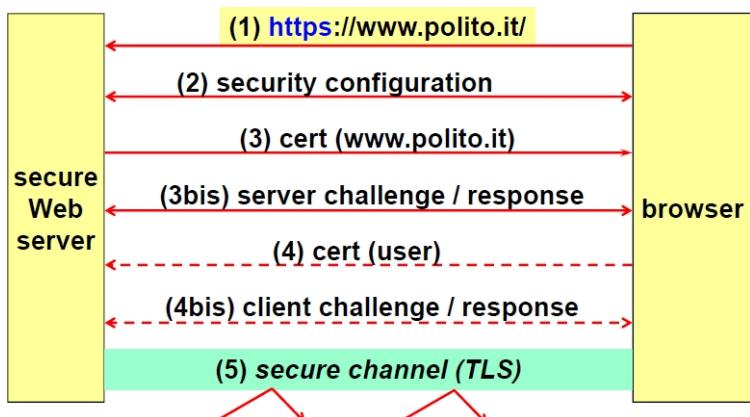
- Un digest con chiave (SHA-1 o migliore) per dimostrare l'autenticazione e l'integrità;
- Un Message Identifier, cioè un numero per evitare risposte e cancellazioni: è possibile perché SSL è stratificato su TCP e non ci sono mai messaggi mancanti, duplicati o fuori ordine all'interno di TCP. Ad esempio, un messaggio di applicazione con lo stesso identificatore di un altro è sicuramente il risultato di un attacco di risposta, perché in caso di pacchetto mancante TCP se ne sarebbe occupato prima di consegnare i dati al livello superiore. Allo stesso modo, se dopo aver ricevuto un messaggio con `id=1` l'applicazione riceve un messaggio con `id=3`, allora sicuramente non si tratta di un pacchetto mancante, ma di un attacco di cancellazione, infatti TCP si sarebbe occupato di ritrasmettere il pacchetto perso (o ritardato).

SSL - Riservatezza

Opzionalmente è possibile avere la riservatezza:

- Il client genera una chiave di sessione utilizzata successivamente per la crittografia simmetrica dei dati (RC4, 3DES, IDEA, AES, ...);
- Lo scambio di chiavi con il server avviene tramite crittografia a chiave pubblica (RSA, Diffie-Hellman o Fortezza- KEA, una variante di Diffie-Hellman utilizzata in passato dal personale militare statunitense).

TLS



Nell'immagine sono presenti un *server Web sicuro* e un *browser*. Il browser vuole aprire un canale sicuro utilizzando HTTPS. Il primo passo consiste nel **concordare una configurazione di sicurezza**. Poiché esistono diversi algoritmi che possono essere utilizzati, il browser e il server dichiareranno il loro elenco di algoritmi e dovranno trovare una corrispondenza. In questa fase, la connessione *potrebbe fallire* (se non si trova un linguaggio comune). **TLS è un protocollo di negoziazione**.

Successivamente, se si sono accordati su un insieme di algoritmi, il server

restituirà il suo certificato che deve contenere il nome corrispondente all'URL collegato. Quindi, il server deve utilizzare la sua chiave privata per rispondere implicitamente a una sfida (nell'immagine c'è una semplificazione, il browser non sta inviando una sfida). Come sarà più chiaro in seguito, la sfida implicita consiste nella decriptazione di una chiave chiamata "pre-master secret", decisa dal client e inviata al server criptata con la chiave pubblica presente nel certificato.

Opzionalmente, il server può chiedere al browser di inviare il proprio certificato X.509 e quindi il server

invierà esplicitamente una sfida al browser che dovrà utilizzare la chiave privata e fornire una risposta. Se tutto va a buon fine, il canale sicuro viene creato.

Punto di vista dell'architettura SSL-3

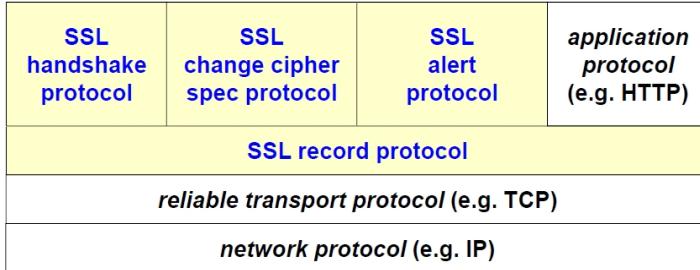


Figura 62 Pila di protocollo parziale quando si usa SSL.

Alla base ci sono il protocollo di rete (ad esempio, IP) e un protocollo di trasporto affidabile (ad esempio, TCP). Sopra il TCP c'è un **protocollo di registrazione** per trasmettere i record SSL (cioè i dati protetti). Al suo interno sono presenti dati applicativi, ma inizialmente viene utilizzato per il **protocollo di handshake** (cioè la fase di accordo dell'immagine precedente). Poi, c'è anche il **protocollo change cipher spec** utilizzato alla fine dell'handshake per comunicare ai sistemi che l'accordo è terminato. Infine, c'è il protocollo

protocollo di avviso, che consente di inviare allarmi (pacchetti mancanti o duplicati) o semplicemente di segnalare la fine della comunicazione.

Quando il protocollo record viene utilizzato per inviare dati applicativi, è necessario inviare file di grandi dimensioni (ad esempio, file di grandi dimensioni dal server) e in questo caso SSL ha un limite: è possibile inviare solo pacchetti di una lunghezza specifica (32 Kbyte). I dati dell'applicazione vengono prima **frammentati** e poi **compressi** (solo nelle versioni più vecchie, perché dava origine ad attacchi) per risparmiare spazio durante la trasmissione. Quindi viene calcolato un MAC per proteggere l'autenticazione e l'integrità. Se si desidera anche la riservatezza, viene aggiunto del padding. Come ultimo passaggio, il nuovo pacchetto compresso+MAC+P viene crittografato e poi viene aggiunta un'intestazione per renderlo riconoscibile come segmento SSL all'interno di TCP.

Formato del record TLS-1.0

- `uint8 type = change_cipher_spec (20), alert (21), handshake (22), dati_applicazione (23)`
- `uint16 version = major (uint8) + minor (uint8)`
- `uint16 lunghezza:`
 - $\leq 2^{14}$ (record non compresso) per la compatibilità con SSL-2
 - $\leq 2^{14} + 1024$ (record compressi)
- Ciò significa che si tratta di un'intestazione di 5 byte.
- Più un carico utile di max 16kB

type	
major	minor
length	
...	
fragment [length]	
...	

SSL - calcolo del MAC

MAC = message_digest (key, seq_number || type || version || length || fragment)

Per proteggere l'autenticazione e l'integrità, viene calcolato un MAC con un hash crittografico specifico (algoritmo digest) e la chiave negoziata durante l'handshake. Tutto è protetto da questo MAC: numero di sequenza, tipo di pacchetto, versione, lunghezza e i dati stessi (il frammento).

Alcune note:

- **Messaggio_digest**
 - Dipende dall'algoritmo scelto;
- **Chiave**
 - È diversa a seconda della direzione: una chiave per proteggere i dati dal client al server; una chiave per proteggere i dati dal server al client. È importante perché altrimenti un

Un aggressore potrebbe copiare un pacchetto inviato dal client al server e rispondere come parte del pacchetto inviato dal server al client.

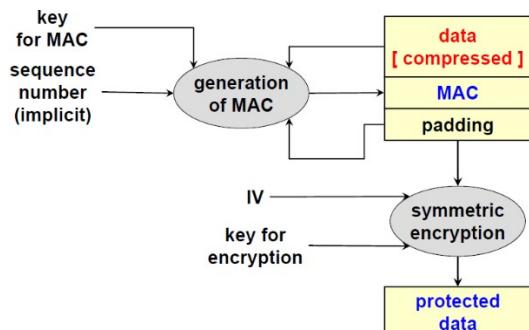
- **Numero_seq**

- Si tratta di un intero a 64 bit che **non viene mai trasmesso**, ma calcolato implicitamente (perché viene utilizzato il protocollo TCP e quindi non si possono perdere pacchetti). Un numero intero così grande permette di mantenere aperto il canale inviando molti dati prima di raggiungere il numero massimo espresso con 64 bit. Quando questo limite viene raggiunto, il canale deve essere chiuso e ne verrà aperto un altro.

Protocollo di handshake SSL/TLS

Il protocollo handshake è necessario per:

- **Concordare un insieme di algoritmi** per la riservatezza e l'integrità;
- **Scambio di numeri casuali** tra il client e il server da **utilizzare per le successive operazioni di generazione delle chiavi**;
- **Stabilire una chiave simmetrica** mediante operazioni a chiave pubblica (RSA, DH o Fortezza);
- **Negoziare l'ID di sessione**: serve a evitare la rinegoziazione dei parametri di sicurezza quando si contatta più volte lo stesso server;
- **Scambiate i certificati necessari**.



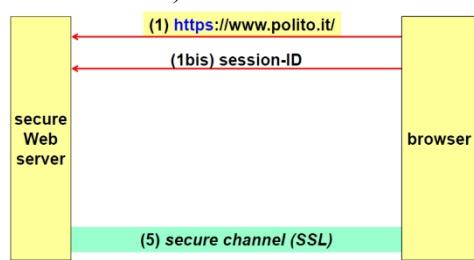
Dal punto di vista della protezione dei dati, ci sono dati (eventualmente compressi) che devono essere protetti. Un MAC viene calcolato prendendo i dati, la chiave, il numero di sequenza隐式的 e l'eventuale padding. Quindi viene inserito il MAC. Il pacchetto viene quindi crittografato simmetricamente con un IV (pensando a CBC) e una chiave specifica per la crittografia. Alla fine, i dati sono protetti e ad essi viene aggiunta l'intestazione SSL.

ID sessione

Una tipica transazione web si presenta come segue:

- 1. Aprire, 2. GET page.htm, 3. page.htm, 4. Chiudere
- 1. Aprire, 2. GET home.gif, 3. home.gif, 4. Chiudere
- 1. Aprire, 2. GET logo.gif, 3. logo.gif, 4. Chiudere
- 1. Aprire, 2. GET back.jpg, 3. back.jpg, 4. Chiudere
- 1. Aprire, 2. GET music.mid, 3. music.mid, 4. Chiudere

I canali vengono continuamente aperti e chiusi! Questo è particolarmente stressante quando si usa l'SSL per aprire i canali, perché se i parametri crittografici dell'SSL devono essere negoziati ogni volta, il carico computazionale (operazioni a chiave pubblica) diventa molto elevato. Per evitare di rinegoziare tutti i parametri crittografici per ogni connessione SSL, il server SSL può inviare un identificatore permanente chiamato **identificatore di sessione** (più connessioni possono far parte della stessa sessione logica): si tratta di un modo per rappresentare un insieme di algoritmi e chiavi che sono stati negoziati in un certo momento. Se il client, quando apre la connessione SSL, prima di iniziare l'handshake, invia un **session-id** valido (già negoziato in precedenza), la parte di negoziazione viene saltata e i dati vengono immediatamente scambiati sul canale sicuro. Si noti che il server può rifiutare l'uso del session-id (sempre o dopo un certo tempo dalla sua emissione).

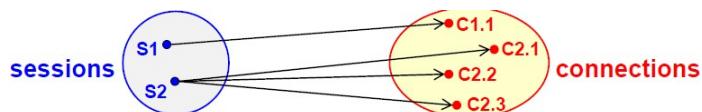


Nella figura, prima di iniziare la negoziazione, viene inviato immediatamente un ID di sessione. Se l'ID di sessione è ancora presente nella cache del server, il canale sicuro viene aperto. Se un attaccante tenta di utilizzare lo stesso ID di sessione valido, non

sarà in grado di inviare messaggi attraverso il canale perché non sa quali algoritmi/chiavi sono stati utilizzati e non può calcolare il MAC corretto. Tuttavia, il tempo per mantenere la sessione-

L'ID con tutte le informazioni correlate deve essere scelto con cura per evitare che il server sia sovraccaricato dal tenere traccia delle sessioni degli utenti. Quindi, ci sono due concetti da tenere a mente quando si tratta di SSL:

- **Sessione TLS**
 - È un'associazione logica tra client e server.
 - Creato dal protocollo Handshake
 - Definisce un insieme di parametri crittografici (algoritmi, chiavi e così via...)
 - È condiviso da una o più connessioni SSL (1: N)
- **Connessione TLS**
 - Un canale TLS transitorio tra client e server
 - Associato a una specifica sessione TLS (1:1)



Relazione tra chiavi e sessioni (tra un server e lo stesso client)

Quando il protocollo di handshake è in esecuzione, vengono stabilite due chiavi pubbliche, chiamate **pre-master secret** (segreto a lungo termine) e **master secret**. Quando la connessione è la prima della sessione, viene generato il segreto pre-master. Quindi, per ogni connessione, compresa la prima, il client e il server generano un **numero casuale** che viene combinato con il pre-master secret per creare il **master secret** (una sorta di operazione simmetrica, come la funzione di derivazione della chiave). Per ogni connessione,

le chiavi per il MAC, le chiavi per la crittografia e l'IV per la crittografia sono ricavate combinando le chiavi per il MAC, le chiavi per la crittografia e l'IV per la crittografia.

Il master secret con i valori casuali che sono diversi per ogni connessione e vengono scambiati in chiaro. L'unica cosa persistente nella stessa sessione è il master secret: la prima volta sarà generato a partire dal pre-segretario master e dai numeri casuali che vengono generati ogni volta che c'è una nuova connessione.

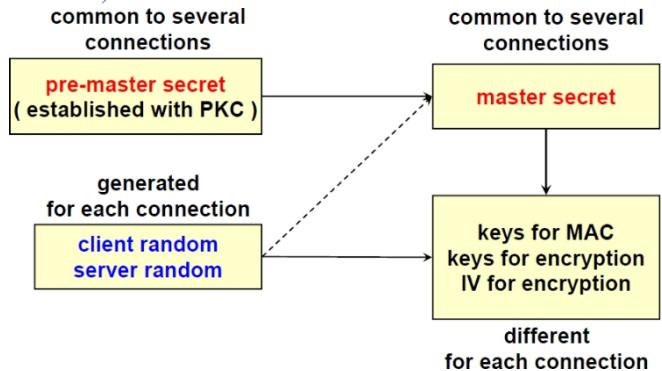


Figura 64 Questa figura si riferisce alla generazione di chiavi e segreti, senza i meccanismi aggiuntivi per ottenere la perfetta segretezza in avanti.

Perfetta segretezza in avanti

Se un server TLS dispone di un certificato valido sia per la firma che per la crittografia, può essere utilizzato sia per l'autenticazione (tramite firma) che per lo scambio di chiavi (crittografia asimmetrica della chiave di sessione). Ma se un aggressore copia tutto il traffico cifrato e in seguito scopre la chiave privata a lungo termine, può decifrare tutto il traffico: passato, presente e futuro. Per evitare questo scenario si ricorre alla "perfetta segretezza in avanti": in questo scenario la compromissione di una chiave privata compromette solo il traffico attuale (ed eventualmente quello futuro) ma non quello passato.

Ciò implica che la chiave utilizzata per l'autenticazione deve essere diversa da quella utilizzata per la crittografia, ma non è sufficiente. La chiave utilizzata per la crittografia deve essere una chiave unica.

"Meccanismi effimeri"

Oggi, per implementare il PFS, la crittografia prevede una chiave unica generata al volo:

- **Per l'autenticità, la chiave deve essere firmata** (ma non può avere un certificato X.509 associato, perché il **processo di CA è lento** e spesso non in linea).
- DH adatto, RSA lento (perché coinvolge numeri primi)
 - Il compromesso per utilizzare una chiave RSA effimera consiste nel generarla una volta e

riutilizzarla N volte (in genere 10-100 volte).

Generando una chiave di crittografia, che viene generata al volo, e firmandola con la chiave privata, si ha il vantaggio che la chiave privata del server viene utilizzata solo per la firma, ottenendo così una perfetta segretezza in avanti:

- Se la chiave privata (temporanea o di breve durata) viene compromessa, l'aggressore può decifrare solo il traffico relativo;
- La compromissione della chiave privata a lungo termine è un problema **per l'autenticazione, ma non per la riservatezza.**

Esempi: I server TLS oggi utilizzano *ECDHE* (*Elliptic Curve Diffie Hellmann Ephemeral*) che prevede che i parametri Diffie Hellmann non siano fissi ma generati al volo.

TLS e server virtuali: il problema

TLS è un canale sicuro attivato prima del livello di applicazione. Questo crea problemi soprattutto con i server virtuali (caso frequente nel web hosting). Data la scarsità di indirizzi IP, è prassi normale creare un server virtuale che è un server con un nome diverso ma con lo stesso indirizzo di un altro: ad esempio, *home.myweb.it=1.2.3.4, food.myweb.it=1.2.3.4*.

È facile farlo in HTTP/1.1, perché il client apre un canale TCP verso quell'indirizzo e poi invia l'intestazione Host per specificare il nome del server a cui vuole connettersi.

Ma è difficile da applicare in HTTPS perché TLS viene attivato prima di HTTP. Quando il browser apre la porta TCP 443 verso l'indirizzo 1.2.3.4, quale certificato deve fornire il server? In effetti, il nome del certificato deve corrispondere al nome digitato come URL.

TLS e server virtuali: soluzioni

Una soluzione è quella di creare un **certificato collettivo** (*wildcard*) come *CN=*.myweb.it*. I problemi sono che la chiave privata deve essere condivisa tra tutti i server che condividono quel certificato e che non tutti i browser trattano questo tipo di certificati speciali allo stesso modo (cioè, l'effettiva efficacia dipende dal browser dell'utente).

Oltre al nome comune, è possibile inserire un "**nome alternativo del soggetto**" (*subjectAltName*) che è un elenco di server virtuali che condividono lo stesso indirizzo IP. Ciò implica comunque che la chiave privata deve essere **condivisa** da tutti i server e che è necessario **riemettere il certificato** a ogni aggiunta o cancellazione di un server.

La soluzione migliore sarebbe quella di utilizzare l'**estensione SNI** (*Server Name Indication*) in ClientHello (primo messaggio inviato dal client al server) standardizzata da RFC-4366. Questa estensione consente al client di indicare che si conserverà a un nome di server specifico. Purtroppo, trattandosi di un'estensione, il suo supporto da parte dei browser e dei server è limitato.

Estensione ALPN (Application-Layer Protocol Negotiation)

Un altro problema è legato al fatto che una volta aperto il canale TLS, c'è un protocollo applicativo che gira protetto al suo interno. Ma anche i protocolli applicativi possono avere diverse versioni e questo è un problema. Questo problema è risolto dall'**estensione ALPN** (RFC-7301).

ALPN è un Application Protocol Negotiation (per TLS-then-proto) per accelerare la creazione della connessione, evitando ulteriori roundtrip per la negoziazione dell'applicazione:

- *ClientHello* afferma *ALPN=true+elenco dei protocolli app. supportati*
- *ServerHello* afferma *ALPN=true+protocollo app. selezionato*

Quando i client selezionano una versione del protocollo da utilizzare, il servizio potrebbe non supportare quel protocollo: in questo caso, la connessione viene chiusa e il client deve riprovare con un'altra versione (magari molte volte). ALPN è un modo per evitare di aprire molti canali TLS e concordare la versione del protocollo applicativo da utilizzare.

È particolarmente importante per negoziare HTTP/2 e QUIC (che è una versione di HTTP basata su UDP), perché Chrome e Firefox supportano HTTP/2 **solo su TLS**. È utile anche per quei server che utilizzano certificati diversi per i vari protocolli applicativi.

Alcuni valori possibili sono http/1.0, http/1.1, h2 (sta per http/2), h2c.

DTLS

Visto il successo di TLS, si è cercato di utilizzare la stessa idea per proteggere anche i datagrammi (poiché TLS può essere utilizzato solo su protocolli affidabili di livello 4, ad esempio TCP). **DTLS** è **Datagram Transport Layer Security** (RFC-4347) e applica i concetti di TLS alla sicurezza dei datagrammi (ad esempio, UDP), ma non offre le stesse proprietà di TLS (ad esempio, cancellazione dei dati, replay dei dati perché nel livello 3 i pacchetti possono essere persi o duplicati).

DTLS è in concorrenza con IPsec e con la sicurezza delle applicazioni, quindi ci sono alcune scelte da fare quando si implementa la sicurezza. Ad esempio, parliamo della sicurezza di **SIP** (protocollo utilizzato per il VoIP). È possibile implementare la sicurezza su SIP:

- Con IPsec (poiché normalmente SIP utilizza pacchetti UDP che vengono trasportati su IP)
- Con TLS (solo per SIP_over_TCP)
- Con DTLS (solo per SIP_over_UDP)
- Con SIP sicuro (esegue la protezione direttamente a livello di

applicazione) Molto spesso la decisione è di tipo pratico: quale dei due è più facile da implementare?

Il problema del downgrade TLS

Esistono varie versioni di TLS: l'SSL originale, la versione 2, 3 e poi TLS 1.0, 1.1, 1.2, ... In genere le versioni più vecchie hanno problemi di sicurezza e non dovrebbero essere utilizzate se non esplicitamente richiesto.

Normalmente, il client invia nel messaggio ClientHello la versione più alta supportata e il server notifica (in ServerHello) la versione da utilizzare, che è la più alta in comune con il client.

Una normale negoziazione di versione tra il *client* e il *server* avviene in questo modo:

- **Accordo su TLS-1.2**
 - ($C \rightarrow S$) 3,3 (significa versione maggiore 3, versione minore 3 che di default è TLS-1.2)
 - ($S \rightarrow C$) 3,3 (il server concorda con la versione proposta dal client)
- **Fallback a TLS-1.1 (ad esempio, nessun TLS-1.2 sul server)**
 - ($C \rightarrow S$) 3,3
 - ($S \rightarrow C$) 3,2 (il server passa a una versione precedente)

Qui c'è un problema: alcuni server, invece di inviare la risposta corretta, chiudono la connessione; a quel punto il client non ha altra scelta che riprovare con una versione di protocollo inferiore. Questo porta all'attacco Downgrade: l'attaccante invia *risposte false al server*, per forzare ripetuti downgrade fino a raggiungere una versione vulnerabile (ad esempio, SSL-3) e quindi eseguire un attacco adeguato (ad esempio, Poodle). Ma non sono possibili solo attacchi, perché potrebbe verificarsi, ad esempio, un errore e la connessione con il server viene chiusa a causa di un problema di rete. Per questo motivo è stata aggiunta una nuova estensione.

TLS fallback Valore della suite di cifratura di segnalazione (SCSV)

Questa estensione è riportata nella RFC-7507 per prevenire gli attacchi di downgrade del protocollo. Ciò avviene attraverso la creazione di una nuova suite di cifratura (fittizia) che non elenca alcun tipo di algoritmo, ma menziona solo TLS_FALLBACK_SCSV che DOVREBBE essere inviata dal client all'apertura di una connessione declassata, ponendola come ultima nell'elenco delle suite di cifratura.

Esempio: supponiamo che il cliente abbia proposto la versione 1.1, ma che sia stato effettuato un downgrade alla 1.0. La risposta sarà inviata con 1.0, ma poiché è possibile fare di meglio, è possibile segnalarlo inserendo FALLBACK_SCSV nella suite di cifratura.

Se è possibile utilizzare una versione migliore, il server invia un nuovo valore di avviso fatale "inappropriate_fallback". Deve essere inviato dal server quando riceve TLS_FALLBACK_SCSV e una

versione inferiore a quella più alta supportata. Se c'è stato un errore, il canale viene chiuso e il client deve riprovare con la versione di protocollo più alta.



Molti server non supportano ancora CSV, ma la maggior parte dei server ha corretto il proprio comportamento scorretto quando il client richiede una versione superiore a quella supportata, quindi i browser possono ora disabilitare il downgrade insicuro: Firefox dal 2015 e Chrome dal 2016.

Sicurezza HTTP

Oltre a TLS c'è spesso HTTP. In HTTP/1.0 sono stati definiti due meccanismi di sicurezza:

- "address-based": il server esegue il **controllo degli accessi** in base all'indirizzo IP del client (inutile a causa dell'IP spoofing);
- "password-based" (o *Basic Authentication Scheme*): controllo dell'accesso basato solo su nome utente e password codificati Base64 (quasi in chiaro, perché la codifica non è una crittografia e non è necessario fornire una chiave)

Entrambi gli schemi sono altamente insicuri (poiché HTTP presuppone che siano utilizzati su un canale sicuro!). In HTTP/1.1 è stata introdotta l'"**autenticazione digest**", basata su un protocollo simmetrico di risposta alla sfida, documentato in RFC-2617 "*HTTP authentication: basic and digest access authentication*".

HTTP - Autenticazione di base

```
GET /path/to/protected/page/ HTTP/1.0
HTTP/1.0 401 Unauthorized - authentication failed
WWW-Authenticate: Basic realm="POLITO - didattica"
Authorization: Basic czEyMzQ1NjptZWdyZXRpc3NpbWE=
HTTP/1.0 200 OK
Server: Apache/1.3
Content-type: text/html
<html> ... protected page ... </html>
$ echo czEyMzQ1NjptZWdyZXRpc3NpbWE= | openssl enc -a -d
s123456:Segretissima
```

Il client sta inviando al server una richiesta (ad esempio, GET di una pagina) con HTTP/1.0. Una pagina protetta ha lo stesso aspetto di una non protetta. Il client riceve un errore *Unauthorized - authentication failed*. L'errore 401 è l'unico in cui il server è autorizzato a non chiudere il canale, ma a mantenerlo aperto. Il motivo è che il client non sapeva che la pagina richiedeva l'autenticazione. Il server continua il dialogo con una nuova intestazione "WWW-

"*Authenticate*", che indica al client che per ottenere la pagina è necessaria un'autenticazione (con meccanismo di base) sul **realm** "POLITO - didattica", che in questo caso è un suggerimento che verrà visualizzato in un messaggio pop-up per suggerire al client quali username e password utilizzare.

Quando il browser riceve questo messaggio, apre un pop-up per fornire nome utente/password per aprire la pagina. Dopo che l'utente ha inserito le credenziali, il browser invia il nuovo comando "*Authorization: Basic czEyMzQ1NjptZWdyZXRpc3NpbWE=*" che è la codifica Base64 di nome utente e password. Il server confronterà questa stringa con il nome utente e la password memorizzati nella sua memoria e, se sono validi, fornirà la pagina originariamente richiesta.

Poiché è solo codificato, è possibile elaborarlo con openssl senza richiedere alcuna chiave. Chiunque potrebbe ottenere le credenziali (a meno che non si trovi all'interno di un canale crittografato).

Autenticazione HTTP digest

HTTP 1.1 ha aggiunto un nuovo modo di autenticare il client, riportato in RFC-2069 (teoricamente obsoleto, ma è ancora il caso base in RFC-2617). La sfida è implicita e la risposta deve utilizzare la password.

Il calcolo del digest è il seguente:

- Il browser calcola un digest: $HA1 = md5(A1) = md5(user ":" realm ":" pwd);$
- Quindi il browser calcola un altro digest: $HA2 = md5(A2) = md5(\text{metodo di accesso (ad esempio, } GET\text{)} ":" URL);$

- Infine il browser invia al server: $response = md5(HA1 ":" nonce ":" HA2)$
 - Il nonce è generato dal server e viene inviato nella risposta 401: il suo scopo è quello di evitare gli attacchi replay;

- Il server di autenticazione può inserire un campo "opaco" per **trasportare informazioni sullo stato** (ad esempio, un token [SAML](#)) verso il server dei contenuti.

```
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" pwd=antonio
HTTP/1.1 200 OK
Server: NCSA/1.3
Content-type: text/html
<HTML> pagina protetta ... </HTML>
```

In questo caso, il GET è realizzato con il protocollo HTTP/1.1. Il server ha due opzioni: l'autenticazione può essere effettuata con la base o con il digest. In questo caso, il server restituisce l'*errore 401* e poi propone l'autenticazione digest fornendo *il realm, il nonce e l'opaque*.

Il browser chiederà nome utente e password e poi invierà la stringa blu riportata nell'immagine. Il server risponderà con la pagina richiesta.

HTTP e SSL/TLS

Esistono due approcci diversi per combinare HTTP con SSL/TLS:

- "TLS then HTTP"* (*RFC-2818 - HTTP over TLS*): viene aperto un canale TCP, su di esso viene creato TLS e quando TLS è attivo si avvia HTTP;
- "HTTP then TLS"* (*RFC-2817 - aggiornamento a TLS all'interno di HTTP/1.1*): due porte TCP 80 possono essere collegate e iniziare a parlare HTTP, quando ci sono dati che richiedono protezione un canale normale può essere trasformato in uno sicuro. Ciò avviene con un comando speciale (*start TLS*) che consente ai browser e ai server di trasformare il canale da normale a TLS.
- Nota: "SSL then HTTP" è molto diffuso, ma non è documentato.*

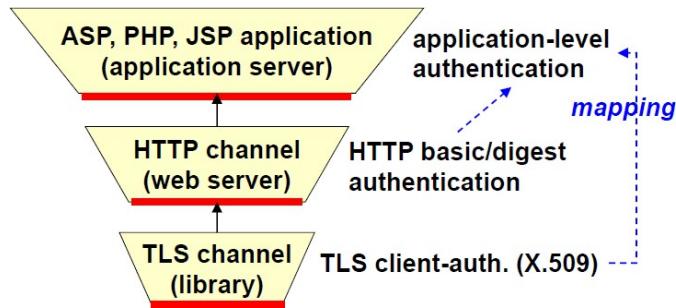
I due approcci principali non sono equivalenti in termini di risultato finale (cioè, non sarà lo stesso), ma ovviamente entrambi convergono verso un canale sicuro, oltre ad avere un impatto su: applicazioni, firewall e IDS (*Intrusion Detection System*).

- Applicazioni:** se si dispone di TLS e poi di HTTP, lo sviluppatore dell'applicazione non ha problemi perché TLS è gestito dal gestore del sistema del nodo o dal gestore della rete e lo sviluppatore web sa che è protetto e non ha problemi al riguardo. In questo caso TLS è sempre attivo, anche per le pagine che non richiedono TLS.
Al contrario, con HTTP e poi TLS, lo sviluppatore web ha la possibilità di attivare TLS quando necessario, ma è lo sviluppatore dell'applicazione che si occupa di questa parte e deve capire cosa sta facendo e gestirla correttamente.
- Firewall:** è un elemento che filtra il traffico (decide cosa può passare e cosa deve essere bloccato). Se si utilizza TLS e poi HTTP, significa che ci sono due porte diverse: la porta TCP 80 per HTTP, la porta TCP 443 per HTTP su TLS. Il firewall può distinguere facilmente il traffico: il traffico sulla porta 80 non è protetto, quello sulla porta 443 è protetto.
Al contrario, se c'è HTTP e poi TLS, il firewall non può fare alcun controllo perché viene usata solo la porta 80 per HTTP e poi lo sviluppatore può decidere di trasformarla da canale semplice a canale protetto. Il firewall non ha più il compito di selezionare solo i canali sicuri.
- IDS:** è un sistema che ispeziona il traffico. Nel caso dell'IDS, se viene utilizzato TLS e poi HTTP, l'IDS non è in grado di ispezionare nulla, perché il traffico è crittografato.
Al contrario, se viene utilizzato HTTP e poi TLS, l'IDS può almeno dare un'occhiata alla parte iniziale e vedere quando il canale viene trasformato in un canale sicuro.

Autenticazione client TLS a livello di applicazione

Tramite l'autenticazione del client è possibile identificare l'utente che ha aperto il canale TLS (senza chiedere nome utente e password). Alcuni server web supportano una mappatura (semi)automatica tra le credenziali estratte dal certificato X.509 e gli utenti del servizio HTTP e/o del sistema operativo. Quando l'autenticazione viene eseguita a questo livello, viene riconosciuta automaticamente anche a livello di applicazione.

Autenticazione nelle applicazioni web



Quando c'è un'applicazione web stratificata su TLS, ci sono diversi livelli, riportati in figura, in cui l'autenticazione può essere eseguita. Il punto importante è che quanto più precoce è il controllo degli accessi, tanto minore è la superficie di attacco.

Se il TLS è abilitato, la prima parte utilizzata per la connessione è la **libreria TLS** (che è piuttosto piccola, implementa solo il TLS) e la piccola parte rossa è quella che può essere attaccata.

Se l'autenticazione del client TLS è attivata, significa che se l'attaccante non dispone di un certificato X.509 o non è autorizzato all'accesso, l'attacco verrà respinto a quel livello e l'unica parte del sistema che può essere attaccata è la libreria TLS stessa. Se TLS è attivato ma non c'è autenticazione del client, allora la libreria TLS può essere attaccata, ma anche il server web stesso (perché prima di accedere all'applicazione web è necessario HTTP, ma un server HTTP è un software piuttosto grande, che presenta più bug di una libreria TLS). È possibile eseguire l'autenticazione a questo livello con *l'autenticazione di base/digest* e se l'attaccante fallisce a questo livello non procede (e il canale viene chiuso). Se l'HTTP è implementato senza autenticazione, si raggiunge l'*application server*, ovvero la pagina che lo sviluppatore ha scritto (ASP, PHP, JSP) con i moduli (username/password richiesti all'utente). A questo livello è troppo tardi, perché l'aggressore può tentare di utilizzare bug nella libreria TLS, nel server HTTP e bug nelle pagine dell'applicazione. Per questo motivo, l'autenticazione dovrebbe essere eseguita il prima possibile per ridurre la superficie di attacco. Attenzione: c'è sempre la possibilità di eseguire una mappatura dall'autenticazione TLS o dall'autenticazione HTTP e portarla nelle pagine dell'applicazione (*esempio del Prof. Lioy eseguito sul sito web di PoliTo nella Lezione 16 del 20th Nov 2020*).

E i moduli che richiedono user/password?

Tecnicamente parlando, non è importante che la pagina contenente il modulo sia sicura (ad esempio, `http://www.ecomm.it/login.html`), perché la sicurezza effettiva dipende dall'URL del metodo utilizzato per inviare nome utente e password al server (ad esempio, `<form ... action=https://www.ecomm.it/login.php>`).

Se la pagina non è protetta, l'utente potrebbe essere esposto al **phishing**, perché altre persone potrebbero creare una pagina web falsa con un indirizzo URL simile, ma con un'altra azione eseguita nel modulo e solo pochi utenti hanno le conoscenze tecniche per verificare l'URL del metodo HTTP utilizzato per inviare utente/password. Se non si utilizza il protocollo HTTPS, non c'è alcuna prova che ci si trovi sul sito web corretto.

Sicurezza del trasporto HTTP (HSTS)

Nell'HTTP è stata inserita una nuova intestazione, riportata nell'RFC-6797 e denominata **HSTS**. Tramite questa intestazione, il server HTTP dichiara che tutte le sue interazioni con l'UA (user agent) devono avvenire esclusivamente tramite HTTPS.

Caratteristiche:

- Impedisce il downgrade del protocollo e il dirottamento dei cookie
- È valido solo nella risposta HTTPS

- L'intestazione ha una scadenza che viene rinnovata a ogni accesso.
- Può includere sottodomini (consigliato)
- Può essere precaricato (ad esempio, i browser possono già sapere che quando si contatta un certo dominio è necessario utilizzare TLS):
 - È un po' pericoloso perché se HTTPS fallisce non c'è possibilità di contattare il server.

- Elenco di precaricamento gestito da Google e utilizzato da molti browser = <https://hstspreload.org/>

HSTS: sintassi ed esempi

```
Strict-Transport-Security:
  max-age = <expire-time-in-seconds>
  [ ; includeSubDomains ]
  [ ; preload ]

$ curl -s -D- https://www.paypal.com/ | fgrep -i strict
strict-transport-security: max-age=63072000
$ curl -s -D- https://accounts.google.com/ | grep -i strict
strict-transport-security: max-age=31536000;
includeSubDomains
```

Il server HTTP come risposta può inserire la sintassi riportata nell'immagine ("*Strict-Transport-Security*") che indica il tempo di scadenza e, facoltativamente, include sottodomini e azioni di precaricamento.

Nella seconda parte dell'immagine, viene controllato se qualcuno sta utilizzando HSTS. Il primo esempio è per PayPal e solo la stringa "strict" viene grep. La risposta è quella riportata

nell'immagine, il che significa che per molto tempo PayPal utilizzerà solo TLS. Il secondo esempio è stato eseguito su Google. La risposta è la stessa, ma comprende anche i sottodomini.

HTTP Public Key Pinning (HPKP)

Per evitare che qualcuno crei un certificato falso per un sito web, il gestore del sito può inserire nell'intestazione l'hash della chiave pubblica che sta utilizzando; l'UA (browser) memorizza questa chiave e rifiuta di connettersi a un sito con una chiave diversa, anche se quel sito presenta un certificato valido. Questo è necessario perché in passato l'autorità di certificazione è stata indotta a creare un certificato per un altro server (RFC-7469).

Si tratta di una tecnica TOFU (*Trust On First Use*), pericolosa quando si perde il controllo della chiave o quando la chiave deve essere aggiornata (includere sempre almeno una chiave di backup/secondaria). L'intestazione consente anche di specificare un URI dove i browser possono segnalare le violazioni. L'intestazione può essere utilizzata in modalità **enforcing** o di **sola segnalazione**.

- **Enforcing:** se un server non possiede la chiave pubblica, la connessione viene chiusa.
- **Solo segnalazione:** se il server ha una chiave pubblica diversa, la connessione procede ma viene segnalata all'utente.

La sintassi è riportata a destra. È presente il calcolo della base64-sha256 della chiave pubblica che si vuole correggere ed è possibile specificare il tempo di scadenza, i sottodomini e il report-URI.

Per utilizzare la funzione Report-only e non Enforcing, è necessario utilizzare la seconda intestazione dell'immagine. La sintassi è la stessa.

```
Public-Key-Pins:
  pin-sha256 = " <base64-sha256-of-public-key> ";
  max-age = <expireTime-in-seconds>
  [ ; includeSubDomains]
  [ ; report-uri = " <reportURI> "]
```

```
Public-Key-Pins-Report-Only:
  pin-sha256 = " <base64-sha256-of-public-key> ";
  max-age = <expireTime-in-seconds>
  [ ; includeSubDomains]
  [ ; report-uri = " <reportURI> "]
```

```
$ curl -s -D - https://scotthelme.co.uk/
  | fgrep -i public-key
public-key-pins:
pin-sha256="9dNiZZueNZmyaf3pTkXxDgOzLkjKvI+Nza0ACF5IDwg=";
pin-sha256="X3pGTSOUjeEVw989IJ/cEtXUEmy52zs1TZQrU06KUKg=";
pin-sha256="V+J+71HvE6X0pqGKVgLtxuvk+0f+xowyr3obtq8tbSw=";
pin-sha256="91BW+k9EF6yy9413/fPiHnqy5Ok4UI5sBpBTuOaa/U=";
pin-sha256="ipMu2Xu72A086/35thucbjlfrPaSjuw4HIjSWsxqkb8=";
pin-sha256="+5JdLySIa9rS6xJM+2KHN9CatGKln78GjnDpf4WmI3g=";
pin-sha256="MwICxyqG2b5RbmYFQu1llhQvYZ3mjZghXTRn9BL9q1o=";
includeSubDomains; max-age=2592000;
report-uri="https://scotthelme.report-uri.com/r/d/hpkp/
  enforce"
public-key-pins-report-only:
pin-sha256="X3pGTSOUjeEVw989IJ/cEtXUEmy52zs1TZQrU06KUKg=";
pin-sha256="Vjs2r4z+80wjNcr1YKepWQboSIRi63WsWXhIMN+eWys=";
pin-sha256="IQBnNBEiFuhj+8x6X8XLgh01V9Ic5/V3IRQLNFFc7v4=";
max-age=2592000;
report-uri="https://scotthelme.report-uri.com/r/d/hpkp/
  reportOnly"
```

A sinistra è riportato un esempio di sito web di un blog sulla sicurezza, che utilizza diverse chiavi pubbliche per il backup. Include sottodomini e include il report-URI per la modalità Enforce.

Vi è poi una seconda sezione, con solo tre chiavi

e un URI diverso per segnalare il problema quando non è obbligatorio.

Sistemi di pagamento elettronico

Le motivazioni per i protocolli di pagamento dedicati sono:

- Fallimento del contante digitale, per problemi tecnici e politici (ad esempio, il fallimento del DigiCash);
- Fallimento di un protocollo di pagamento dedicato (SET, Secure Electronic Transactions) a causa di problemi tecnici e organizzativi;
- Attualmente l'approccio più utilizzato è la trasmissione del numero di carta di credito su un canale TLS, ma questo non è una garanzia contro le frodi: negli anni scorsi VISA Europe ha dichiarato che le transazioni via Internet generano circa il 50% dei tentativi di frode, anche se all'epoca rappresentavano solo il 2% dell'importo totale delle transazioni!
 - Ciò significa che Internet non è il luogo in cui vengono rubati i numeri di carta di credito, ma è il luogo in cui è più facile utilizzare un numero di carta di credito rubato;
 - Il fatto che TLS protegga la trasmissione del numero di carta di credito non è una garanzia, perché il problema è legato al fatto che i dati sono stati rubati in precedenza (e utilizzati in Internet) o che il server non è attendibile.

Un'architettura di pagamento basata sul web

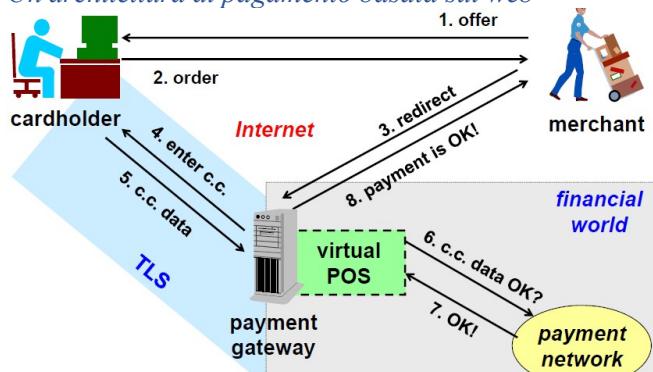


Figura 65 L'architettura più comune nell'implementazione del pagamento con carta di credito in Internet

La prima interazione in questo schema è l'offerta da parte dell'esercente: immaginate che il titolare della carta abbia navigato nel negozio, riempito un carrello e ora sia pronto a effettuare il check-out. Quando il titolare della carta effettua l'ordine, l'impostazione consigliata è che il commerciante non richieda i dati relativi al pagamento, perché è troppo rischioso e richiederebbe di garantire standard di sicurezza molto elevati. La soluzione migliore è che l'esercente abbia stipulato un accordo con un gateway di pagamento e che reindirizzi la richiesta verso di esso. A questo punto il titolare della carta non è più in contatto con l'esercente, ma con il gateway di pagamento, che crea un canale sicuro, attraverso il quale il titolare della carta può essere in contatto con l'esercente.

che vengono richiesti e rispediti i dati della carta di credito. Il problema è quello di convalidare questi dati: se si trattasse di un pagamento fisico, sarebbe stato utilizzato un dispositivo **POS** (*Point Of Sale*), per cui in questo caso il gateway di pagamento crea un **POS virtuale** che, partendo dai dati della carta di credito, crea una transazione valida per la rete di pagamento, per verificare se i dati della carta di credito sono validi per quello specifico pagamento. Quando la rete di pagamento fornisce una risposta positiva a tale controllo, il gateway fornirà una risposta positiva all'esercente, che sarà quindi in grado di fornire al titolare della carta la merce richiesta.

Quindi, per riassumere, la linea di base di questa architettura è:

- l'acquirente possiede una carta di credito;
- l'acquirente dispone di un browser abilitato a TLS.

Tuttavia, la sicurezza effettiva dipende dalla configurazione del server (gateway di pagamento) e del client: TLS è un protocollo di negoziazione, quindi l'importante è il **livello di sicurezza effettivo** che viene negoziato tra il gateway di sicurezza e il browser dell'utente. Inoltre, il gateway di pagamento dispone di tutte le informazioni (pagamento + merce), mentre il commerciante conosce solo le informazioni sulla merce: questa è una buona pratica, poiché è bene fornire all'utente l'elenco della merce che sta acquistando quando inserisce il numero di carta di credito, ma ha lo svantaggio che il gateway potrebbe profilarlo e non rispettare la sua privacy.

Nel caso in cui l'esercente non voglia affidarsi a questa architettura suggerita, ma preferisca eseguire il pagamento autonomamente, allora dovrà conformarsi allo standard **PCI DSS**.

PCI DSS

PCI DSS è l'acronimo di Payment Card Industry Data Security Standard ed è richiesto da tutti gli emittenti di carte di credito per le transazioni via Internet. Si tratta di un insieme di prescrizioni tecniche molto dettagliate rispetto ad altri standard di sicurezza (ad esempio, HIPAA = Health Insurance Portability and Accountability Act).

Si tratta di uno standard piuttosto complesso, quindi la maggior parte delle aziende preferisce non implementarlo da sola, ma affidarsi al gateway di sicurezza. Ci sono eccezioni degne di nota, come Amazon.

Ci sono state alcune versioni, con alcuni aggiornamenti:

- **v2.0 = ottobre 2010;**
- **v3.0 = novembre 2013;**
- **v3.1 = aprile 2015 (senza SSL o "vecchio" TLS);**
- **v3.2 = aprile 2016 (MFA, funzioni/procedure di test, descrizione**

dell'architettura, ...). Le prescrizioni di questo standard sono:

- **progettare, costruire e gestire una rete protetta:**
 - 1st requisito: installare e mantenere una configurazione con firewall per proteggere l'accesso esterno ai dati dei titolari di carta;
 - 2nd requisito: non utilizzare password di sistema predefinite o altri parametri di sicurezza impostati dal produttore.
- **proteggere i dati dei titolari di carta:**
 - 3rd requisito: proteggere i dati dei titolari di carta memorizzati. È possibile criptare i dati quando vengono memorizzati su disco o fornire un forte controllo degli accessi, in modo che solo il personale autorizzato possa leggere i dati del titolare della carta. Lo standard non impone una soluzione, l'importante è poter dimostrare di avere una protezione adeguata;
 - 4th requisito: criptare i dati dei titolari di carta quando vengono trasmessi su una rete pubblica aperta. È obbligatorio.
- **stabilire e seguire un programma per la gestione delle vulnerabilità:**
 - 5th requisito: utilizzare un antivirus e aggiornarlo regolarmente;
 - 6th requisito: sviluppare e mantenere applicazioni e sistemi protetti: implica la dimostrazione che la sicurezza è stata presa in considerazione fin dall'inizio dello sviluppo e che vi è una continua verifica e manutenzione della sicurezza.
- **implementare un forte controllo degli accessi:**
 - 7th requisito: limitare l'accesso ai dati dei titolari di carta solo a quelli necessari per un compito specifico;
 - 8th requisito: assegnare un unico ID a ogni utente. Ciò significa che, all'interno dell'azienda, ogni utente deve avere un unico nome utente, non sono ammessi sistemi di autenticazione multipli. Il motivo di questo requisito è che se ci sono più nomi utente associati allo stesso utente diventa molto difficile assegnare o rimuovere i permessi e tracciare le azioni.
 - 9th requisito: limitare l'accesso fisico ai dati dei titolari di carta.
- **monitorare e testare regolarmente le reti:**
 - 10th requisito: monitorare e tracciare tutti gli accessi alle risorse di rete e ai dati dei titolari di carta. Ciò significa che, come minimo, sono necessari file di log sicuri che consentano di sapere chi ha avuto accesso a quale server e a quali dati;
 - 11th requisito: testare periodicamente i sistemi e le procedure di protezione;
- **adottare una politica di sicurezza:**
 - 12th requisito: adottare una politica di sicurezza, vale a dire che deve esistere un piano di sicurezza ben definito e non solo un tentativo di implementare alcune funzioni di sicurezza aggiornate solo dopo gli attacchi riusciti!

Firewall e IDS/ISP

Che cos'è un firewall

Un firewall è un "muro di protezione contro la propagazione del fuoco": questo concetto deriva da una norma diffusa nei Paesi in cui le case sono costruite prevalentemente in legno, in base alla quale, in determinate condizioni, tra due case deve essere interposto un muro di mattoni per evitare la propagazione del fuoco. Analogamente, dal punto di vista logico, nella sicurezza dei sistemi informativi un firewall è qualcosa che viene posizionato in modo tale che, una volta attaccata una rete (che è "in fiamme"), l'attacco non si propaghi facilmente a un'altra rete ad essa collegata:

È il caso di due reti con livelli di sicurezza diversi, una delle quali non è attendibile e un attacco ad essa potrebbe propagarsi all'altra più sicura.

Quindi, oltre al fatto che il firewall deve essere posto al confine di reti con diversi livelli di sicurezza, un altro aspetto importante da considerare è la direzionalità del flusso da controllare:

- **firewall di ingresso:**
 - È un filtro che controlla le **connessioni in entrata** dalla rete non attendibile;
 - Lo scopo è tipicamente quello di selezionare i servizi (pubblici) offerti; limita quali servizi interni sono offerti alla rete esterna.
 - Il problema è che a volte la richiesta di apertura di una connessione fa parte di uno scambio di applicazioni avviato da utenti interni;
- **firewall di uscita:**
 - È un filtro che controlla le **connessioni in uscita**;
 - Lo scopo è tipicamente quello di controllare l'attività del personale: ciò può includere il controllo dei siti web visitati, per verificare che i documenti riservati non vengano inviati all'esterno della rete o per verificare che un utente non stia scaricando qualche file pericoloso.

Questa classificazione è facile per i servizi basati sui canali (ad esempio, le applicazioni TCP), ma difficile per i servizi basati sui messaggi (ad esempio, le applicazioni ICMP, UDP): in questi casi, infatti, il firewall è tipicamente bidirezionale, e una distinzione tra ingresso e uscita non avrebbe senso.

Progettazione del firewall

Quando si ha bisogno di un firewall, un aspetto da considerare è che il firewall non è un singolo oggetto, ma è fatto di componenti, è un sistema: "**non si "compra" un firewall, lo si progetta (si possono comprare i suoi componenti)!**". Il progetto può essere costituito da un singolo componente, e per questo motivo le aziende dicono di offrire firewall: in realtà, offrono singoli componenti, e tipicamente per costruire un firewall forte è necessario comporne molti. La configurazione dei componenti è tale da permettere di ottenere un compromesso ottimale tra sicurezza e funzionalità, con costi minimi.

Sicurezza, funzionalità e i tre comandamenti del firewall

Quando si progetta un firewall, la prima cosa da considerare è il livello di sicurezza desiderato. Il problema è che esiste una relazione inversa con il livello di funzionalità offerto: maggiori sono le funzionalità offerte. Quindi, tutte le soluzioni sono un compromesso tra questi due obiettivi. Inoltre, nel progettare un firewall si dovrebbe sempre seguire lo stesso principio esplicitato dagli inventori del firewall (*D. Cheswick e S. Bellovin*):

- il FW deve essere **l'unico punto di contatto** della rete interna con quella esterna;
- **solo il traffico "autorizzato" può attraversare il FW**: questo implica avere a priori un elenco preciso del tipo di traffico, di protocollo o di porte o anche di nodi esterni a cui consentire l'accesso,

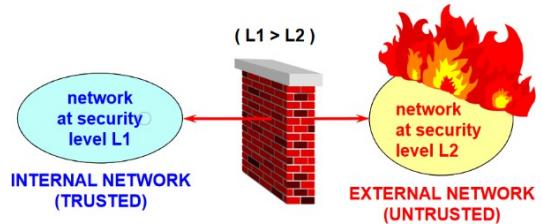


Figura 66 Rappresentazione tipica di un filtro di rete (o protezione di confine)

con il rischio di tagliare fuori il traffico valido (bloccando così le funzionalità) o di essere troppo laschi e aprire la porta ad attacchi;

- **il FW deve essere esso stesso un sistema altamente sicuro:** la maggior parte delle aziende utilizza un potente computer di uso generale per caricarvi il software per il firewall, ma quando si rendono conto che la maggior parte del tempo la macchina è inattiva, iniziano a caricarvi altri componenti (ad esempio, server web, server di database). Questo deve essere evitato perché più software viene installato, più alta è la possibilità di avere bug che possono essere sfruttati per eseguire un attacco.

Politiche di autorizzazione

Per il secondo principio, nella progettazione di un firewall è necessario identificare il traffico autorizzato. Nell'esprimere la politica di autorizzazione, esistono due possibili strategie:

- **whitelisting:** "Tutto ciò che non è esplicitamente permesso, è proibito".
 - il firewall apre alcuni tipi di comunicazione, garantendo così una maggior sicurezza;
 - più difficile da gestire, soprattutto se gli utenti erano abituati a disporre di connettività gratuita. Ciò potrebbe comportare la necessità di fornire una buona dose di spiegazioni su cose che non sono più consentite.
 - È quello suggerito;
- **lista nera:** "Tutto ciò che non è esplicitamente vietato, è permesso".
 - implica lo studio di ciò che può essere un problema di sicurezza e il divieto di quel tipo di traffico, e in genere porta a una minore sicurezza (cancelli aperti), perché è come dire che la vulnerabilità viene mantenuta finché non viene scoperta;
 - più facile da gestire.

Componenti di base di un firewall

Come detto, un firewall è un'architettura composta da diversi componenti:

- **router di schermatura (choke):** router che filtra il traffico a livello di rete;
- **bastion host:** sistema sicuro, con auditing (periodico). È la prima linea di difesa contro gli attacchi;
- **application gateway (proxy):** servizio che lavora per conto di un'applicazione per effettuare comunicazioni con l'esterno della rete, con *controllo degli accessi*;
- **dual-homed gateway:** sistema con due schede di rete, che funge da ponte tra due reti diverse, ma con il normale routing disabilitato: si devono inserire al suo interno alcuni componenti in grado di decidere se il traffico può essere tranquillamente inoltrato o deve essere bloccato.

A seconda del livello di stack di rete considerato da un firewall, vengono utilizzati termini diversi per riferirsi ad esso:

- **packet filter:** significa che il livello di rete è quello utilizzato per analizzare il traffico, e tipicamente vengono sfruttate solo le informazioni contenute in quel livello;
- **gateway di circuito:** lavora a livello 4, quindi considera un flusso TCP o un datagramma UDP come unità da considerare per eseguire il filtraggio;
- **gateway applicativo:** esamina in dettaglio l'applicazione dati;

Di solito, più si sale nello stack, più il rilevamento degli attacchi è accurato, ma più lento.

"Architettura "Screening Router"

Il concetto centrale è quello di dotare l'elemento di commutazione di capacità di filtraggio: tipicamente, trattandosi di un dispositivo di rete L3, il tipo di filtraggio che può effettuare è a livello IP e superiore, implementando quindi un *filtro dei pacchetti*. Il vantaggio di questa architettura è che non c'è bisogno di hardware dedicato;

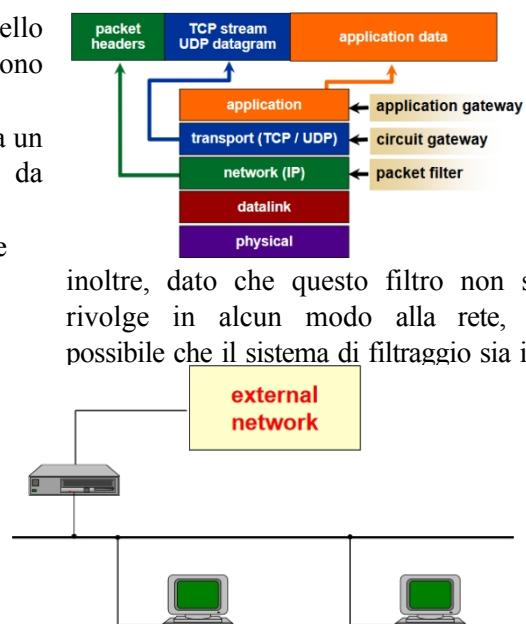


Figura 68 Architettura del router di screening

Figura 67 A quale livello vengono effettuati i controlli?

a livello di applicazione, non c'è bisogno di un proxy e quindi di modificare le applicazioni. È una soluzione semplice, facile, economica e... *insicura*, poiché il tipo di controllo che può effettuare è molto banale, di solito basato su protocolli, porte e indirizzi. Un ultimo svantaggio è che il router è un singolo punto di guasto, il che significa che un bug potrebbe far sì che l'attaccante aggiri il controllo e acceda alla rete interna.

"Architettura "dual-homed gateway"

Poiché il solo filtraggio a livello di rete non garantisce un'elevata sicurezza, per migliorare l'architettura è necessario implementare un altro elemento che esegua ispezioni a livelli superiori. In questa architettura, il traffico consentito dal filtro dei pacchetti non entrerà direttamente nella rete interna, ma sarà ulteriormente controllato da un secondo elemento, generalmente chiamato "gateway", che in realtà è un "dual-homed" perché ha due schede di rete, con il routing disabilitato dal momento che una

Il pacchetto proveniente da un'interfaccia va verso l'altro se rispetta le regole imposte.

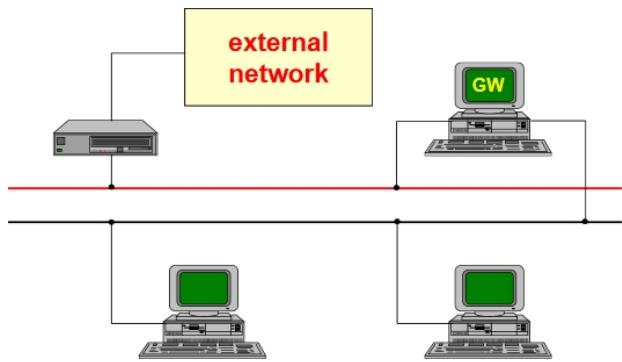


Figura 69 Architettura del gateway dual-homed

È comunque facile da implementare, perché si tratta di due componenti diversi che si occupano di aspetti separati. Richiede pochi requisiti hardware aggiuntivi, basta una macchina generica con il software gateway appropriato installato. Inoltre, poiché il gateway è il frontend dell'intera rete interna, può mascherare gli indirizzi interni, anche senza utilizzare un NAT.

Il grosso problema è che questa soluzione è piuttosto inflessibile, perché anche se un traffico dovrebbe essere consentito nel filtro dei pacchetti, poi deve anche passare il controllo al gateway: prendendo come esempio la posta elettronica, tipicamente nella posta in arrivo è difficile limitare chi è il mittente (a parte vietare i siti noti che inviano spam), quindi viene controllato al server di posta, che è interno alla rete. Il gateway ricostruirebbe il pacchetto a livello di applicazione solo per vedere che si tratta di una mail e che non è suo compito ispezionarla, quindi la invierà alla rete interna: l'inflessibilità sta nel fatto che un pacchetto viene controllato due volte anche se il secondo controllo è inutile perché non si può decidere nulla con le informazioni che ha il gateway. Le azioni svolte dal gateway comportano un grande overhead di lavoro. Una cosa da notare è che questa soluzione implementa una **doppia linea di difesa**, quindi è un'applicazione del principio della "*difesa in profondità*".

"Architettura "ospite schermato"

Migliorare l'architettura "dual-homed gateway" significa evitare il collo di bottiglia del gateway: nell'architettura "screened host", il filtro dei pacchetti è collegato anche alla rete interna, ma c'è anche il gateway, che è collegato al filtro dei pacchetti, in modo che i pacchetti che non necessitano di un doppio controllo vengano inoltrati direttamente alla rete interna. In questo modo si ripropone il problema del singolo punto di guasto nel filtro dei pacchetti e si implementa una doppia linea di difesa solo per i pacchetti che devono essere elaborati in modo più approfondito.

Quindi, per riassumere:

- il **router**:

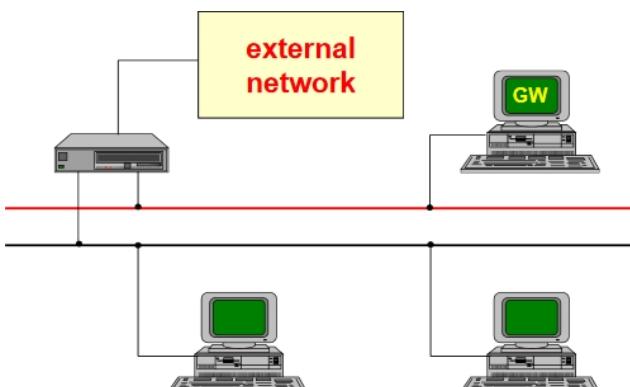


Figura 70 Architettura host schermata

- blocca il traffico INT \leftarrow EXT a meno che il traffico non provenga dal bastione;
- blocca il traffico EXT \leftarrow INT a meno che il traffico non vada al bastione;
- eccezione: servizi abilitati direttamente.
- L'**host bastione** gestisce il gateway di circuito/applicazione per controllare i servizi autorizzati;

- più costoso e complesso da gestire, perché il lavoro dei due sistemi deve essere sincronizzato;
- più flessibile (nessun controllo su alcuni servizi/ospiti);
- solo gli host/protocolli che passano attraverso il bastione possono essere mascherati (a meno che il router non usi NAT).

"Architettura della "sottorete schermata"

Il problema principale introdotto con l'architettura "host schermato" è il singolo punto di fallimento del filtro dei pacchetti. Nell'architettura "screened subnet", il filtro dei pacchetti è diviso, perché esegue concettualmente due operazioni diverse:

- decidere se un traffico in entrata deve essere consentito o negato;
- decidere se il traffico deve essere inoltrato alla rete interna.

In questa soluzione, un filtro dei pacchetti è collegato alla rete esterna, mentre l'altro agisce come un ponte tra la rete "intermedia" e quella interna. In primo luogo un pacchetto in arrivo viene controllato se è consentito, e se è consentito direttamente, viene inoltrato al secondo filtro dei pacchetti che verificherà ancora se la regola è valida e poi inoltrerà alla rete interna: in questo modo c'è una doppia difesa anche per i pacchetti che saranno trasmessi direttamente alla rete interna. Per i pacchetti che necessitano di ulteriori ispezioni, ci saranno tre linee di difesa, poiché il secondo filtro dei pacchetti li invierà a

il bastione. Si noti che, per avere una doppia linea di difesa adeguata implementata dai due router,

dovrebbero essere venduti da fornitori diversi, perché se sono uguali forse anche i bug sono gli stessi, e questo influenza sulla sicurezza.

In questo schema c'è una rete completamente disaccoppiata sia dalla rete esterna che da quella interna (quella in rosso nella figura): si chiama **DMZ** (*De-Militarized Zone*). In genere, la DMZ ospita non solo il gateway ma anche altri host (tipicamente i server pubblici, come il server web o i sistemi di accesso remoto) e questo significa che i server pubblici dovranno essere collocati nella DMZ. Questa soluzione è più costosa delle altre, perché è più complessa.

Architettura "sottorete schermata" (versione 2)

Per ridurre i costi e semplificare la gestione, spesso i router vengono omessi e la loro funzione è incorporata nel gateway: questa soluzione è chiamata anche "**firewall a tre gambe**". In questa soluzione il firewall è un singolo elemento, tipicamente un computer di uso generale, dotato di tre reti

schede: una si collega alla rete esterna, una a quella interna e la terza implementa la DMZ.

I componenti della soluzione precedente sono moduli software all'interno di un singolo elemento, quindi da un punto di vista funzionale ci sono le stesse funzionalità, ma la soluzione è intrinsecamente meno sicura, perché:

- Anche in questo caso c'è un singolo punto di guasto, la singola macchina che ospita tutti i processi;
- La complessità di questo nodo lascia spazio ad attacchi dovuti a errori di implementazione o di configurazione.

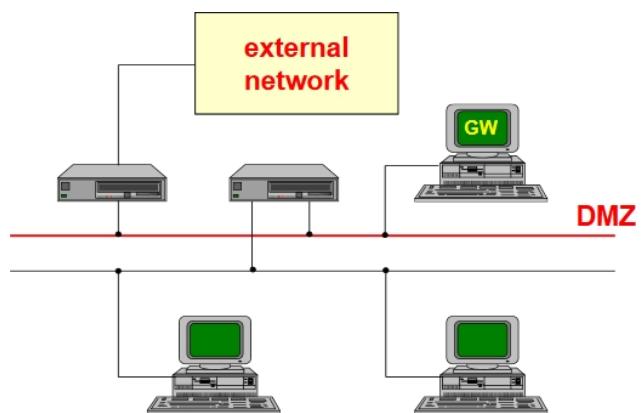


Figura 71 Architettura della sottorete schermata

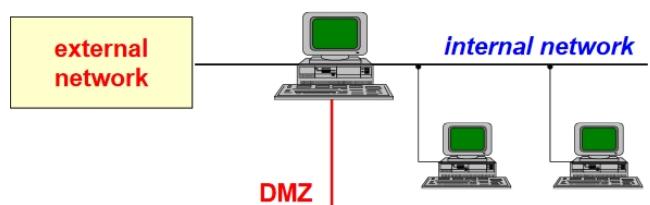


Figura 72 Architettura della sottorete schermata (versione 2)

Nonostante queste considerazioni, molte aziende tendono a implementare questa soluzione perché offre molte funzionalità a fronte di un costo limitato, e la gestione è semplice perché c'è solo un'interfaccia per gestire tutto. Alcune aziende affermano di aver implementato un "*firewall a quattro gambe*" o "*a cinque gambe*": il numero di gambe deriva dall'avere più interfacce verso la rete interna o da

Se si distribuiscono più DMZ, ognuna può essere assegnata a un reparto diverso dell'azienda, soprattutto per le decisioni amministrative.

Tecnologie firewall

A seconda del livello di rete in cui vengono eseguiti i controlli, esistono termini diversi:

- *Filtro pacchetti (statico)*
- *Filtro pacchetti Stateful (dinamico)*
- *Proxy di cut-off*
- *Gateway/proxy a livello di circuito*
- *Gateway/proxy a livello di applicazione*
- *Ispezione stateful*

Le differenze sono in termini di:

- *Controlli da eseguire (=minacce rilevate):*
 - Sono relativi alle minacce che possono essere rilevate
- *Prestazioni:*
 - È importante comprendere le diverse tecnologie che possono fornire diversi livelli di prestazioni, in genere perché la sicurezza ha un prezzo.
- *Protezione del firewall O.S.:*
 - Diverse soluzioni prevedono una diversa protezione del firewall stesso
- *Mantenere o rompere il modello client-server:*
 - I firewall di cui si parla sono posti tra il client e il server, quindi possono essere trasparenti, il che significa che se il traffico è consentito il client può parlare direttamente con il server finale, oppure il firewall può agire come un proxy, in modo che il client parli con il firewall e poi parli con il server. Se il firewall interrompe il modello client-server, la soluzione è più sicura, ma il firewall stesso diventa un elemento soggetto ad attacchi;

(Statico) Filtro a pacchetti

Originariamente era disponibile sui router ed eseguiva l'ispezione dei pacchetti a livello di rete controllando solo l'intestazione IP e, quando disponibile, anche l'intestazione di trasporto. Il motivo è che una volta ricevuto un pacchetto, questo deve essere inoltrato o scartato nel minor tempo possibile.

Questo tipo di soluzione ha pro e contro:

- **È indipendente dal tipo di applicazioni**
 - *Buona scalabilità (se il throughput aumenta)*
 - *Controlli approssimativi: facili da "ingannare" (ad esempio, spoofing IP, pacchetti frammentati)*
- **Buone prestazioni**
- **Basso costo (disponibile sui router e in molti sistemi operativi)**
- **Difficile supportare servizi con porte allocate dinamicamente (ad es. FTP)**
- **Complesso da configurare:** tutte le regole sono espresse in termini di IP, porte e protocolli;
- **Difficoltà nell'eseguire l'autenticazione dell'utente:** deriva dal fatto che utilizza solo le informazioni disponibili a L3;

Filtro pacchetti Stateful (dinamico)

- **Concettualmente simile al filtro dei pacchetti, ma "state-aware":** significa che *ricorda i pacchetti precedenti per essere più veloce nell'elaborazione dei nuovi*. Può cercare all'interno dei pacchetti i comandi che definiscono le porte da utilizzare (ad esempio, il comando FTP PORT). **È in grado di distinguere le nuove connessioni da quelle già aperte:**
 - Mantiene le tabelle di stato per le connessioni aperte;
 - I pacchetti che corrispondono a una riga della tabella vengono inoltrati senza ulteriori controlli (trasmissione veloce);

- **Prestazioni migliori rispetto al filtro dei pacchetti:**
 - È possibile attivare il supporto SMP (*Symmetrical Multi-Processing*), grazie all'uso di tabelle di stato;
- **Presenta ancora molte delle limitazioni dei filtri statici dei pacchetti;**

Gateway a livello di applicazione

È composto da una serie di proxy che ispezionano il payload dei pacchetti a livello di applicazione. Il gateway spesso **richiede modifiche all'applicazione client** e può opzionalmente **mascherare/rinumerare gli indirizzi IP interni**. Quando viene utilizzato come parte di un firewall, di solito esegue anche l'**autenticazione tra pari** (in genere per il firewall di uscita). Poiché il proxy comprende i comandi a livello di applicazione, può fornire **una sicurezza superiore**, ad esempio contro l'overflow del buffer dell'applicazione di destinazione in caso di attacco. Esistono differenze tra *forward-proxy* (egress) e *reverse-proxy* (ingress).

In generale, quando si lavora a livello di applicazione, le regole possono essere più **fini e più semplici** di quelle di un filtro di pacchetti, perché è possibile esprimere le regole in termini di comandi e dati a livello di applicazione.

Ogni applicazione ha bisogno di un proxy specifico perché avrà i propri comandi e dati, il che significa:

- *Ritardo nel supporto di nuove applicazioni;*
- *Pesante per le risorse computazionali*: per ogni connessione è necessario un processo separato, che in genere viene eseguito in modalità utente;
- *Prestazioni ridotte (perché sono processi in modalità utente);*

È possibile utilizzare SMP che può migliorare le prestazioni. Questo **rompe** completamente **il modello client/server**, il che significa:

- *Maggiore protezione per il server*
- *Può autenticare il cliente*
- *Non è trasparente per il client*: l'applicazione deve essere configurata per utilizzare il proxy;

Poiché non è trasparente, significa che il sistema operativo del firewall può essere esposto ad attacchi, perché deve elaborare tutti i pacchetti. Inoltre, c'è anche il problema delle tecniche di sicurezza a livello di applicazione (ad esempio, SSL) che non consentono di ispezionare il traffico. Per questo motivo, esistono alcune varianti:

- **Proxy trasparente**
 - Meno invasivo per il cliente
 - Più lavoro (reinstradamento dei pacchetti + estrazione della destinazione)
- **Forte proxy dell'applicazione** (*verifica della semantica, non solo della sintassi*)
 - Solo alcuni comandi/dati vengono inoltrati, ad esempio nel caso del protocollo HTTP, questo potrebbe consentire solo i comandi GET e POST;
 - Questa è l'unica configurazione corretta per un proxy (secondo Lioy)

Gateway a livello di circuito

Tra il filtro dei pacchetti e il gateway a livello di applicazione c'è anche il **livello di circuito**. Si tratta di un **proxy generico** (che non è "application-aware") che **crea un circuito a livello di trasporto** tra client e server, ma non comprende né manipola in alcun modo i dati del payload. Si limita a copiare tra le sue due interfacce i segmenti TCP o i datagrammi UDP (se corrispondono alle regole di controllo dell'accesso) ma, nel fare questo, riassembla i pacchetti IP e quindi fornisce una protezione contro alcuni attacchi L3/L4. Per il server, tutti gli attacchi legati all'handshake TCP non sono più possibili perché non c'è alcun handshake TCP tra client e server. Il client esegue l'handshake TCP con il gateway e poi il gateway esegue un handshake corretto con il server.

In conclusione, interrompe il modello client/server a livello TCP/UDP durante la connessione e fornisce:

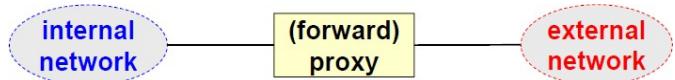
- **Maggiore protezione per il server:**

- Isolato da tutti gli attacchi relativi all'handshake TCP;
- Isolato da tutti gli attacchi legati alla frammentazione IP: un aggressore potrebbe frammentare un pacchetto con cui esegue l'attacco, e tale attacco non sarà rilevato da un semplice filtro dei pacchetti;
- **Può autenticare il client:**
 - Ma questo richiede la modifica dell'applicazione;

Presenta ancora molte limitazioni del filtro dei pacchetti. Il più famoso è **SOCKS**.

Proxy HTTP (forward)

Il forward proxy, tipicamente HTTP, è un server HTTP che agisce solo come front-end e poi passa le richieste al vero server, che è esterno. I vantaggi, oltre alle **ACL** (Access Control List) di rete:



- *Cache condivisa delle pagine esterne per tutti gli utenti interni;*
- *Autenticazione e autorizzazione degli utenti interni;*
- *Vari controlli (ad esempio, siti consentiti, direzione di trasferimento, tipi di dati, ...);*

È una parte tipica del firewall **di ingresso**. Al contrario, il **reverse proxy** è un firewall di ingresso.

Proxy inverso HTTP

Si tratta di un server HTTP che funge solo da front-end per i server reali a cui vengono passate le richieste. Può implementare ACL nel caso in cui sia possibile limitare i client, ma in genere un firewall di ingresso non limita i client che possono contattare il server. È possibile eseguire l'**ispezione dei contenuti**. I vantaggi di questo proxy sono:

- **Offuscamento** (*nessuna informazione sul server reale*)
 - Poiché il server proxy risponde al client, può dichiarare di essere un proxy generico senza fornire alcuna informazione al software reale che implementa il server finale.
- **Acceleratore SSL** (*con connessioni back-end non protette...*)
 - Il proxy inverso può essere il punto finale per SSL/TLS. In questo senso è un acceleratore SSL perché se il canale viene terminato qui, è possibile collocare un HSM e ottenere il vantaggio aggiuntivo che il traffico arriva in chiaro dopo la terminazione del canale TLS. Questo permette di eseguire l'ispezione dei contenuti.
- **Bilanciatore di carico**
- **Acceleratore web** (=cache per contenuti statici)
- **Compressione**
- **Alimentazione al cucchiaio**
 - Ottiene dal server un'intera pagina dinamica e la trasmette al client in base alla sua velocità, scaricando così il server delle applicazioni;

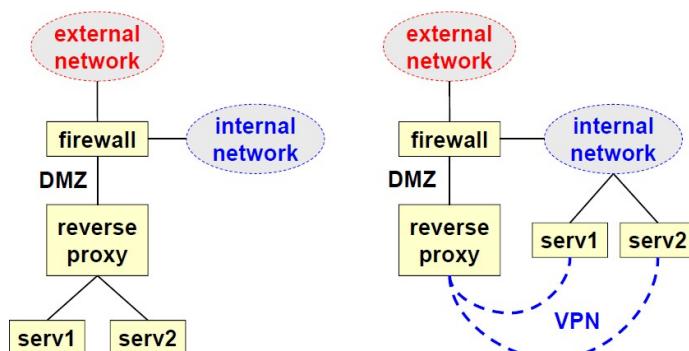


Figura 73 Possibili configurazioni del firewall a tre gambe quando si utilizza un reverse proxy. Quella a sinistra è quella suggerita.

Esistono due possibili configurazioni per il reverse proxy: la soluzione migliore è che, se concettualmente si utilizza un firewall a tre gambe, il reverse proxy venga posizionato sulla DMZ perché sarà l'interfaccia pubblica, e quindi è possibile posizionare dietro di esso i server equivalenti (che sono i server applicativi a cui accedono gli utenti esterni). Il problema si pone nel caso in cui il server applicativo debba accedere a dati situati all'interno della rete interna: in questo caso il server deve passare dal proxy, attraversare il firewall e infine entrare

nella rete interna.

Per gestire questo caso evitando quanto appena spiegato, è possibile anche una soluzione alternativa: il reverse proxy si trova sulla DMZ e poi viene stabilita una connessione VPN tra il reverse proxy e i server, che si trovano nella rete interna. Questo dovrebbe limitare i rischi perché non c'è accesso diretto a quei server, ma solo attraverso il reverse proxy. Tuttavia, la prima soluzione è quella suggerita, poiché in questo caso un attacco contro i server (pubblici) è limitato alla DMZ.

WAF (Web Application Firewall)

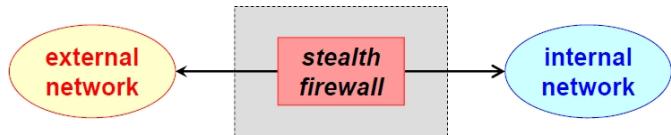
Poiché al giorno d'oggi le applicazioni web sono largamente utilizzate, il numero di minacce è in aumento. Il WAF è un modulo installato in un proxy (forward e/o reverse) per **filtrare il traffico delle applicazioni**. Controlla:

- Comandi HTTP
- Intestazione della richiesta/risposta HTTP
- Contenuto della richiesta/risposta HTTP

Il WAF più conosciuto e utilizzato è **ModSecurity** (progetto opensource) che è un plugin per *Apache* e *NGINX* (che rappresentano il 50% e il 30% dei server HTTP mondiali). ModSecurity è così popolare che OWASP (*Open Web Application Security Project*) ha sviluppato uno specifico **ModSecurity Core Rule Set (CRS)**. Ciò significa che OWASP ha studiato quali sono gli attacchi più frequenti e conosciuti e ha scritto un insieme di regole che permettono a ModSecurity di rilevare e bloccare tali attacchi.

Firewall stealth

Si tratta di un firewall **senza indirizzo IP**, che non può essere attaccato direttamente. Se un firewall non ha un indirizzo IP, c'è il problema che il firewall non può essere il punto finale di una comunicazione. In questo **c a s o**, però, lo stealth



Il firewall intercetta fisicamente i pacchetti impostando le interfacce in **modalità promiscua** (modalità in cui la scheda di rete riceve una copia di tutti i pacchetti che raggiungono l'interfaccia). Quindi, in base a una logica, il compito del firewall sarà quello di **copiare o scartare il traffico di rete** (in base alla sua politica di sicurezza), ma non lo altera in alcun modo. Se il firewall non è indirizzato nella rete non è possibile fare manutenzione/configurazione da remoto, a meno che un'altra scheda di rete non sia attaccata al firewall ma collegata alla rete di gestione della sicurezza dedicata.

Firewall locale/personale

Finora abbiamo parlato di firewall di rete, che è un filtro che si colloca tra due reti (interna ed esterna), ma questo tipo di firewall ha un problema: HTTPS, che significa che i canali sono criptati. È difficile guardare all'interno dei pacchetti.

Questo ha portato a spostare il firewall dove è possibile ispezionare il traffico. In HTTPS il traffico è crittografato sul client e sul server. Ciò significa che il firewall deve essere spostato sul client o sul server: **firewall locale** (se installato sul server) o **personale** (se installato sul client). È **installato direttamente sul nodo da proteggere**. Protegge un singolo nodo anziché un'intera rete. È tipicamente un **filtro dei pacchetti** ma, rispetto a un normale firewall di rete, può **limitare i processi consentiti**:

- *Aprire canali di rete verso altri nodi (cioè, agire come client).*
- *Rispondere alle richieste della rete (cioè, agire come un server).*

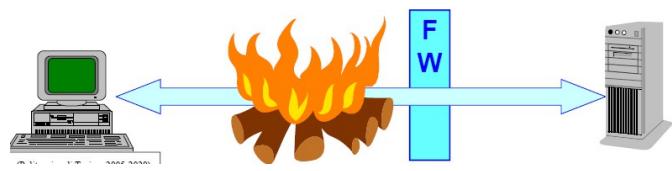
Ad esempio, questo firewall può limitare il traffico verso un processo sconosciuto, perché potrebbe essere un malware sul client che sta cercando di ottenere alcune informazioni.

È importante per limitare la diffusione di malware e trojan, o per rilevare semplici errori di configurazione (ad esempio alcuni servizi installati di default sul sistema operativo che inviano o ricevono dati). La gestione del firewall deve essere separata dalla gestione del sistema, altrimenti chi installa un servizio attiverà anche il

firewall per consentire quel tipo di traffico (che dovrebbe essere deciso dal responsabile della sicurezza).

Protezione offerta da un firewall

L'immagine mostra il fatto che nei punti del muro in cui i mattoni vengono rimossi per consentire il traffico, quella è la parte in cui il fuoco può entrare nel sistema. Per tutti i canali che sono stati abilitati attraverso il firewall, sono necessarie altre protezioni:



- *VPN;*
- *firewall/IDS "semantici";*
- *Sicurezza a livello di applicazione.*

Sistema di rilevamento delle intrusioni (IDS)

L'IDS è un altro componente importante di qualsiasi soluzione di sicurezza al giorno d'oggi: è un **sistema per identificare gli attori che utilizzano un computer o una rete senza autorizzazione**. A seconda delle caratteristiche, può essere esteso per **identificare gli attori autorizzati che violano i loro privilegi**.

L'IDS si basa su un'ipotesi importante: **il "modello" comportamentale degli utenti non autorizzati è diverso da quello degli utenti autorizzati**.

Un IDS può basare il suo comportamento su due strategie:

- **IDS passivo**
 - Cerca di identificare i segni visibili di un attacco, utilizzando:
 - Controllo del *checksum crittografico* (ad esempio, tripwire) per i file modificati sul server che non dovrebbero essere modificati;
 - *Pattern matching* ("firma dell'attacco") alla ricerca di pacchetti specifici tipici di un attacco;
- **IDS attivo**
 - Cerca di identificare attacchi sconosciuti o noti, ma prima che producano un risultato negativo. Il sistema si articola in tre fasi:
 - "*apprendimento*" = analisi statistica accurata del comportamento del sistema;
 - "*monitoraggio*" = raccolta attiva di informazioni statistiche su traffico, dati, sequenze, azioni;
 - "*reazione*" = confronto con parametri statistici (reazione al superamento di una soglia).

Un IDS attivo ha sempre qualcuno che controlla le statistiche dell'IDS. Questo è necessario perché le statistiche possono non essere accurate. Un attacco viene rilevato se si discosta significativamente dalle statistiche relative al comportamento del sistema, quindi c'è la possibilità di avere attacchi non rilevati ma anche falsi positivi. Ciò significa che spesso è necessario l'intervento umano per controllare gli allarmi lanciati dall'IDS, analizzare attentamente il problema e capire se si tratta di un attacco o solo di un falso positivo, data la soglia rigorosa. In genere, gli IDS hanno una parte attiva e una passiva.

Caratteristiche topologiche degli IDS:

- **HIDS (host-based IDS)**, cerca le statistiche all'interno di un singolo nodo:
 - *Analisi dei log (sistema operativo, servizio o applicazione);*
 - *Strumenti di monitoraggio del sistema operativo interno;*
- **I NIDS (IDS basati sulla rete)** analizzano il traffico di rete:
 - *Strumenti di monitoraggio del traffico di rete.*

Componenti HIDS: SIV e LFM

- **Verificatore di integrità del sistema**
 - Controlla i file/filesystem alla ricerca di modifiche non autorizzate (ad esempio, modifiche al registro di Windows, configurazione di cron, privilegi degli utenti).

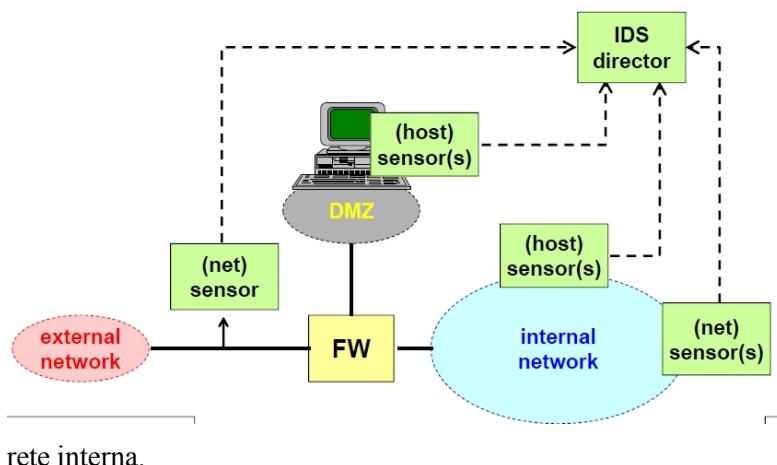
- ad esempio, *tripwire*

- **Monitoraggio dei file di registro**
 - Controlla i file di log (sistema operativo e applicazioni) alla ricerca di modelli noti di attacchi o tentativi riusciti (ad esempio, un numero elevato di accessi non riusciti).
 - ad esempio, *swatch*

Componenti NIDS

- **Sensore** (host/rete):
 - Controlla il traffico e i registri alla ricerca di schemi sospetti;
 - Quindi generare gli eventi di sicurezza pertinenti;
 - Può anche interagire con il sistema (modificare le ACL, eseguire il reset TCP, ...).
- **Direttore:**
 - Coordina il lavoro di tutti i sensori;
 - Gestisce il *database della sicurezza* (ad esempio, statistiche, firme di attacco);
- **Sistema di messaggi IDS** (per far comunicare direttore e sensori):
 - Fornisce una comunicazione sicura (autenticazione e integrità) e affidabile (nessuno deve essere in grado di far cadere un messaggio inviato dai sensori o viceversa) tra i componenti dell'IDS. Per ottenere l'affidabilità, la comunicazione, se possibile, avviene su una rete fisica separata. Se non è possibile, si utilizza una VLAN o una VPN.

Architettura NIDS



Lo schema mostra come dovrebbe essere configurato l'IDS. Concettualmente c'è sempre il firewall a tre gambe con una connessione verso l'esterno, una verso l'interno e una verso la DMZ. Nella rete interna vengono inseriti i **sensori** sugli **host** più critici (ad esempio, database, application server). Ci sono anche **sensori di rete** necessari per verificare se qualche traffico malevolo riesce a passare la rete, ma anche per controllare il comportamento degli utenti nella rete interna.

Nella DMZ sono presenti server raggiungibili dall'esterno e facilmente attaccabili. Pertanto, è necessario inserire un **sensore host** su ogni server posizionato sulla DMZ. L'ultimo sensore è il **sensore di rete sull'interfaccia esterna del firewall**. Qualcuno pensa che non sia necessario perché si trova all'esterno, ma collocarlo all'esterno del firewall permette di analizzare quali sono i tentativi, di ottenere statistiche (ad esempio, il firewall blocca il 98% degli attacchi).

In alto a destra, il direttore IDS raccoglie tutte le informazioni provenienti dai sensori.

IPS - Sistema di prevenzione delle intrusioni

L'IPS è un sistema utilizzato per accelerare e automatizzare la reazione alle intrusioni. È l'accoppiata di un **IDS** e di un **firewall dinamico distribuito**. L'IDS invia allarmi relativi a un certo tipo di traffico, quindi il firewall dinamico distribuito quando riceve un allarme può bloccare tutto il traffico di quel tipo specifico o isolare un nodo. Questo è un bene se il rilevamento del sistema di rilevamento delle intrusioni è sicuro al 100%, ma se non lo è, l'IPS è pericoloso perché potrebbe prendere decisioni sbagliate e bloccare il traffico innocente. Si tratta di una *tecnologia*, non di un prodotto, con un grande impatto su molti elementi del sistema di protezione, che spesso viene integrato in un unico prodotto, chiamato IDPS.

Firewall di nuova generazione (NGFW)

L'NGFW è un tipo di firewall che **cerca di identificare le applicazioni indipendentemente dalla porta di rete utilizzata**. Immaginate che gli utenti possano utilizzare solo HTTPS (443/TCP) mentre tutto il resto del

traffico è bloccato. Potrebbe esserci qualcuno che utilizza la porta 443 per inviare non HTTPS ma altri tipi di traffico. Con un filtro a pacchetti non è possibile

per rilevarlo. Al contrario, con l'NGFW è possibile esaminare il traffico su qualsiasi porta e cercare di capire qual è l'applicazione reale che viene eseguita da quella porta. Se possibile, l'NGFW **cerca di decifrare/ricifrare** il traffico.

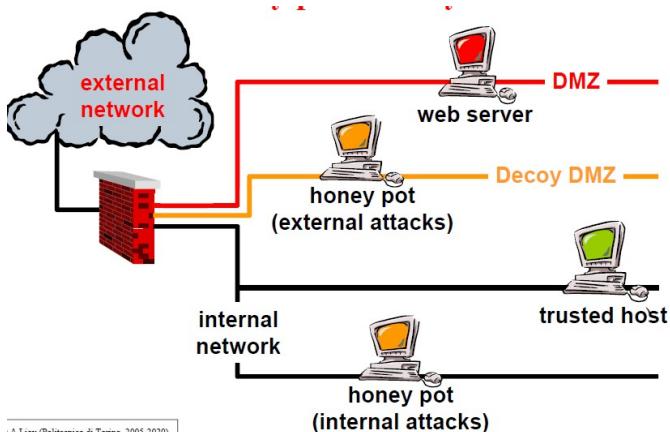
Offre l'integrazione con sistemi di autenticazione come captive portal, 802.1x, o endpoint authN (Kerberos, Active Directory, LDAP, ...), perché in questo modo mentre normalmente i firewall di rete scrivono politiche basate sugli indirizzi IP, se c'è un'integrazione con un sistema che dice a quale utente è assegnato un indirizzo IP, è possibile scrivere **politiche per utente** e non per indirizzi IP. Allo stesso modo, il fatto che sia possibile identificare le applicazioni indipendentemente dalla porta utilizzata, permette di applicare anche politiche **per-applicazione**. Inoltre, è possibile eseguire un filtraggio basato anche su vulnerabilità, minacce e malware noti.

Gestione unificata delle minacce (UTM)

Si tratta di un'**integrazione di diversi prodotti in un unico dispositivo** che ne semplifica la gestione. Il dispositivo fornito è denominato *UTM appliance/Security appliance* che può contenere *firewall, VPN, anti-malware, content-inspection, IDPS, ...*

Le capacità effettive dipendono dal produttore. L'obiettivo principale è quello di ridurre il numero di sistemi diversi, quindi la complessità di gestione e i costi.

Vaso di miele/rete di miele



L'honey pot è un modo per **attirare gli aggressori**. La rete esterna è protetta da un firewall, che contiene la vera DMZ, ma in aggiunta c'è una seconda DMZ, chiamata **decoy DMZ**, dove si trova un **honey pot**. Si tratta di una **macchina facilmente attaccabile**. Lo scopo è quello di attirare gli aggressori e poi avere un sistema di ispezione che monitora ciò che l'aggressore sta facendo, per **capire il comportamento** degli aggressori e di nuovi tipi di attacchi, o per raccogliere malware. Anche i produttori di antimalware hanno piazzato Honey Pot in tutto il mondo proprio per raccogliere nuovi malware.

L'honey pot può essere collocato anche all'interno della rete interna, per rilevare se ci sono utenti interni che cercano di attaccare il sistema. Ad esempio, è possibile creare un falso server di stipendi duplicati per verificare se qualcuno sta cercando di aumentare il proprio stipendio o di leggere quello di qualcun altro.

Sicurezza della posta elettronica

MHS (Sistema di gestione dei messaggi)

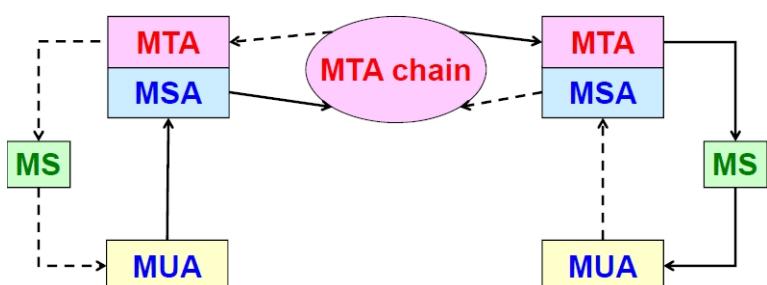


Figura 74 Schema generale del funzionamento del sistema di posta elettronica

Esiste un sistema mondiale che segue l'MHS che contiene diversi componenti elencati nella figura.

Il **MUA (Message User Agent)** è il componente software che l'utente utilizza per inviare la posta. La posta viene poi trasmessa dal MUA all'**MSA (Message Submission Agent)**, che è un altro componente software che ha il compito di iniettare la posta nel trasporto della posta.

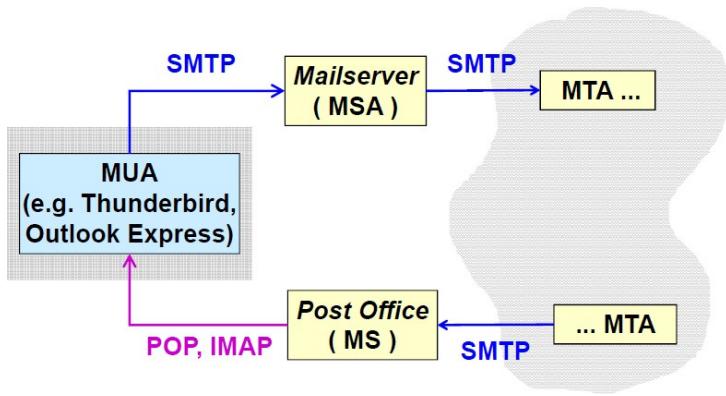
sistema. Il sistema di trasporto della posta è composto da **MTA** (**M**essage **T**ransfer **A**gent) disposti a **catena**: la posta viene consegnata all'MTA successivo e così via (come la consegna della posta negli uffici). La catena termina quando il messaggio viene ricevuto alla destinazione finale, che non è il computer o il dispositivo del singolo utente.

utente, ma è un altro server chiamato **MS (Message Store)**. Infine, la destinazione utilizzerà il MUA per leggere la posta quando lo desidera.

Se la destinazione vuole inviare una risposta, deve seguire un percorso simile, ma l'MSA inverso è diverso da quello di invio, la catena MTA può essere diversa e così via.

Esistono due modi per interfacciarsi con la posta elettronica. Il primo è la **posta elettronica in modalità client-server**.

Posta elettronica in modalità client-server



Significa che l'utente ha un programma specifico (ad esempio, Thunderbird, Outlook) che è configurato per sapere qual è il server di posta o il **server di posta** in uscita (MSA). Utilizzerà il protocollo SMTP per inviare la posta. In questo modo è possibile inviare un messaggio e poi spegnere il dispositivo, ma la posta è stata inviata. Ciò non significa che la posta sia stata ricevuta, perché l'MSA continuerà a inviare quando la rete lo permetterà e quando l'MTA ricevente lo permetterà.

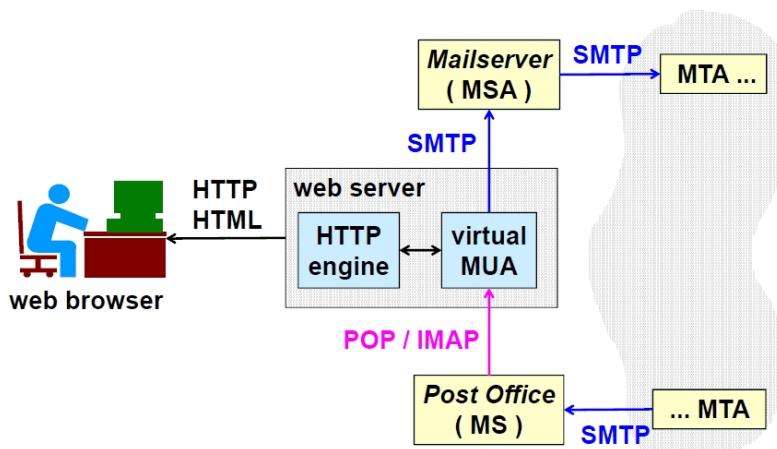
hanno spazio, perché ogni MTA archivia localmente la posta ricevuta e la inoltra all'elemento successivo della catena quando possibile. Anche in caso di **interruzione della rete**, la *posta verrà consegnata. La consegna* della posta può richiedere fino a *tre giorni*.

Quando qualcuno invia un messaggio, lo inietta attraverso la catena MTA e infine il messaggio in arrivo viene memorizzato nel **Post Office (MS, server di posta in arrivo)**.

Si noti che SMTP è un **protocollo push**, in quanto la posta viene sempre inviata all'elemento successivo, ma quando la posta viene ricevuta all'ufficio postale non c'è push, ma un altro protocollo (**POP/IMAP**) viene utilizzato quando un utente vuole controllare la posta elettronica, chiedendo all'ufficio postale i messaggi in arrivo. In questo caso è possibile scegliere se lasciare una copia sulla MS o cancellarla. Dal punto di vista della sicurezza di base, questa è la soluzione migliore: la posta ricevuta viene cancellata dall'ufficio postale. Naturalmente, i vari SM potrebbero aver fatto una copia illegale della posta.

Oggi questo schema non è più utilizzato, il più comune è lo **schema webmail**.

Webmail



Sono coinvolti diversi protocolli, porte e formati:

- **SMTP (Simple Mail Transfer Protocol)**

di posta elettronica.

Protocolli, porte e formati

Lo schema è in realtà lo stesso, con una grande differenza: non c'è un MUA ma un **browser web** (o un'applicazione mobile). Il vero MUA è un **MUA virtuale**, che si trova all'interno del **server web**. Davanti al MUA si trova un **motore HTTP**, che permette di scrivere email con formati web standard (protocollo HTTP, linguaggio HTML, ecc.). Il punto è che l'e-mail viene memorizzata presso il provider del server di posta. Dal punto di vista della privacy *questo non è un bene*. Il motivo per cui c'è uno spazio illimitato per le e-mail è che le aziende hanno il controllo

- 25/TCP (MTA)
- 587/TCP (MSA) durante l'invio di posta elettronica
- **POP (Post Office Protocol)**
 - 110/TCP
- **IMAP (Internet Message Access Protocol)**
 - 143/TCP
- **"RFC-822"**
 - Formato del messaggio (corpo di testo puro).
- **MIME**
 - Estensione multimediale di RFC-822

Messaggi RFC-822

Fornisce messaggi di **testo puro** con solo caratteri US-ASCII su 7 bit, perché gli 8th possono essere usati per un controllo di parità (un residuo di quando le reti erano inaffidabili). Le righe devono essere terminate da <CR> <LF> per rendere le mail indipendenti dal sistema. I messaggi sono composti da intestazione e corpo:

- **Intestazione**
 - Parole chiave all'inizio della riga
 - Le righe di continuazione iniziano con uno spazio
- **Corpo**
 - Separato dall'intestazione da una riga vuota
 - Contiene il messaggio

Intestazione RFC-822

Si noti che anche se un'interfaccia viene utilizzata nella propria lingua, si tratta solo di un'interfaccia perché i veri messaggi inviati in rete hanno intestazioni **scritte in inglese**.

- **From** o **Sender** identificano chi invia il messaggio: **from** è il **mittente logico**, mentre **sender** è il **mittente operativo**. In genere coincidono, ma a volte non è così. Ad esempio, quando si inviano messaggi di posta elettronica con il sistema del Politecnico sono uguali, ma se si utilizza Gmail, c'è un'altra differenza.

■ From:	sender (logical)
■ Sender:	sender (operational)
■ Organization:	organization of the sender
■ To:	destination
■ Subject:	subject
■ Date:	date and hour of sending
■ Received:	intermediate steps
■ Message-Id:	sending ID
■ CC:	copy to
■ Bcc:	copy (hidden) to
■ Return-Receipt-To:	return receipt to

- è l'opzione per inviare la posta con l'account Gmail, ma è possibile etichettarla con un "Da" diverso.
- Poi c'è l'**organizzazione, la destinazione e l'oggetto**.
- **La data** è l'ora dichiarata da MUA, il che significa che è facile falsificare la data in un messaggio.
- **Ricevuto** corrisponde a qualsiasi MTA che è stato attraversato. Qualsiasi MTA vi inserisce informazioni.
- **ID messaggio, CC (copie carbone)**
- **BCC (Blind Carbon Copies)** con destinatario nascosto.
- **Return-Receipt-to** quando il mittente desidera ricevere un ack.

Un esempio SMTP/RFC-822

Poiché SMTP è un protocollo basato sul testo e RFC-822 è un formato basato sul testo, è possibile inviare e-mail con la tastiera utilizzando **telnet**, che apre un terminale basato sul testo con i server remoti.

- **telnet duke.colorado.edu 25**
 - apre una connessione alla porta 25 (che è quella in cui il server è in ascolto)
 - Il server risponde con una risposta positiva (220).
- **HELO Leonardo.polito.it**
 - Tutti i comandi in STMP sono lunghi 4 caratteri, quindi HELO non è un errore.
 - Il server risponde con una risposta positiva
- **MAIL DA: cat**
 - Il server accetta perché non sa chi è dalla parte.
- **RCPT A: franz**
 - Il server di destinazione controlla se si tratta di un utente che ha una casella di posta elettronica su quel sistema. Se la risposta è positiva, significa che *il destinatario è a posto*.
- **DATI**
 - Il server risponde con 354, che è un intermedio positivo.

Quindi, il messaggio inizia a seguire l'RFC-822 e termina con un punto sull'ultima riga, come indicato di seguito.

Il server risponde con *250 Ok*, il che significa che il messaggio è stato inserito nella casella di posta dell'utente *franz@duke.colorado.edu*. Quindi con *QUIT* il canale viene chiuso.

È stato dimostrato che ci sono m o l t e possibilità di creare m
false, perché in "Da" c'è
possibile scrivere qualsiasi cosa. Non viene controllato che il F

telnet duke.colorado.edu 25

Trying

Connected to duke.colorado.edu

Escape character is '^J'

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: vacation

Hello Francesco,

I renew my invitation to come to my place during
your vacation in Italy. Let me know when you arrive.

Antonio

.

250 Ok

QUIT

221 duke.colorado.edu closing connection

connection closed by foreign host

Problemi di sicurezza della posta elettronica

- **Sistema senza connessione** (*store-and-forward*, anche grazie ai record MX)
- **MTA non affidabili**
- **Sicurezza della SM**
 - Poiché il messaggio è memorizzato in MS
- **Crittografia delle mailing list**
 - È necessaria la chiave pubblica della destinazione. Se le e-mail vengono inviate a mailing-list non è possibile crittografarle perché ci sono molte e-mail all'interno.
- **Compatibilità con quanto già installato**
- In passato sono state sviluppate **soluzioni concorrenti**:
 - Internet = PGP, PEM, MOSS, S/MIME
 - OSI = X.400
 - Oggi gli unici due sopravvissuti sono: S/MIME e PGP

Spamming della posta

Chiamata anche **UBE** (*Unsolicited Bulk Email*) o **UCE** (*Unsolicited Commercial E-mail*) è utilizzata per l'invio di messaggi indesiderati:

- **Pubblicità non autorizzata**
- **Attacchi** (*malware, phishing, ...*)

Oggi è quasi il 50% del traffico totale di e-mail (54% a marzo '20, in calo rispetto al 69% del 2012), il che significa:

- Forte carico sui server e sui canali di rete (a causa della sua natura di store and forward)
- Forte fastidio per gli utenti

Il termine "spam" deriva da uno sketch dei Monty Python relativo alla carne di maiale in scatola. A causa dell'origine, il contrario di "spam" è "ham" (termine utilizzato dalle applicazioni di identificazione e filtraggio).

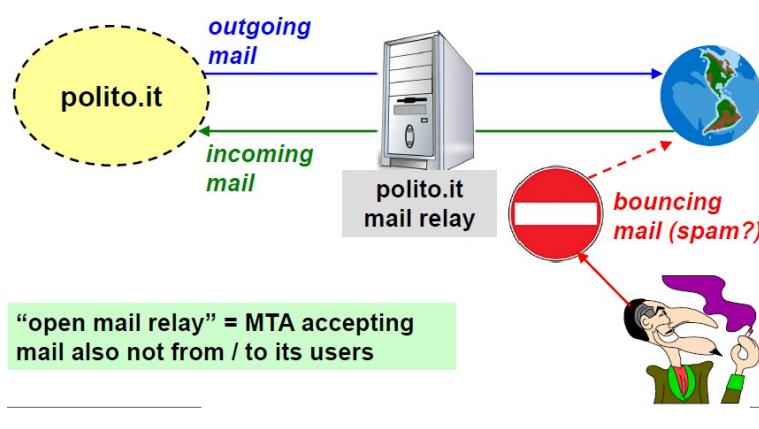
Strategie di spamming

Lo spammer cerca di **nascondere** il vero mittente, utilizzando però un **mittente valido**, perché in questo modo è possibile che la vittima si fidi della posta come se fosse buona. Gli spammer inviano lo spam anche tramite MTA speciali che prendono il nome di **open mail relay** perché sono MTA non ben configurati che accettano e-mail da chiunque e da qualsiasi destinazione. Gli spammer utilizzano anche **zombie o botnet** con **indirizzi IP variabili o fantasma** (utilizzano indirizzi che non dovrebbero essere assegnati).

Un'altra strategia di spamming frequente è l'**offuscamento dei contenuti** (grazie a programmi che controllano il contenuto della posta elettronica per scartare le e-mail contenenti parole chiave associate allo spam):

- **Errori deliberati** (ad esempio, Vi@gr@);
- **Immagine piuttosto che testo** (immagine che contiene testo, in modo da non essere individuata dai software antispamming, che di solito non sono in grado di riconoscere il testo);
- **Avvelenamento bayesiano** (ad esempio, testo tratto da un libro): i sistemi antispam di solito controllano la frequenza delle parole rispetto ai messaggi standard. Per avvelenare questi sistemi di rilevamento, gli spammer inseriscono un breve testo all'inizio, che è il vero messaggio, e poi inseriscono 20-30 righe, ad esempio da Shakespeare.
- **All'interno di un messaggio di errore**: nella posta elettronica non c'è alcuna garanzia che l'e-mail venga inviata a destinazione, e se dopo 3 giorni l'MTA non è in grado di inoltrare il messaggio, invia un errore. Gli spammer inviano quindi falsi messaggi di errore nella speranza che non vengano controllati dai software antispamming.

Relè di posta (aperto)



Normalmente, quando c'è un dominio c'è anche un *domain relay*, composto da due server (o da uno che li comprende entrambi): è configurato per accettare la posta **in uscita** solo se proviene dal dominio "*polito.it*" e accetta la posta in entrata solo se la destinazione finale è "*polito.it*". Al contrario, se c'è qualcuno che invia una mail al mail relay di Polito, ma la destinazione è in qualche altro posto del mondo, la cosa è vietata. Questo significa che il mail relay non è un mail relay aperto.

Se il sistema non è configurato correttamente e queste restrizioni non vengono accettate, ciò che accadrebbe è che un utente esterno potrebbe chiedere al relay di posta Polito di inviare e-mail all'esterno. Tuttavia, questa restrizione deve considerare il caso in cui un utente esterno valido di Polito voglia utilizzare il servizio. Per questo motivo, il mail relay deve essere in grado di distinguere gli utenti reali da quelli sconosciuti o non attendibili, e naturalmente non è possibile utilizzare l'indirizzo IP per identificare se qualcuno proviene dal dominio polito.it.

Anti-spam per MSA

La linea guida è di **non configurare il proprio MSA come un "open relay"**, ma di limitarne l'uso solo agli utenti autorizzati, che devono essere **autenticati**. Per autenticare gli utenti del nostro MSA è possibile verificare che:

- **Indirizzo IP del MUA**
 - È un problema di nodi mobili o di spoofing dell'IP e di malware installato su un nodo valido.
- **Valore del campo Da**
 - Può essere facilmente ingannato con un'email falsa
- **Autenticazione SMTP**
 - È il più sicuro, ma quali sono i metodi di autenticazione sicuri?

Anti-spam per l'MTA in entrata

Elenchi DNSBL

Si può anche identificare lo spamming in entrata, quando si ricevono messaggi di posta elettronica. La strategia si basa sul verificare se l'MTA che sta contattando il MS è un noto spammer, confrontandolo con una blacklist o una whitelist. Una possibilità è quella di controllare una **DNSBL** (*DNS-based BlackList*) che funziona come segue:

- Viene ricevuta una richiesta dall'MTA con indirizzo A.B.C.D.
- L'indirizzo A.B.C.D è utilizzato per inviare spam?
- `nslookup -q=A D.C.B.A.dnsbl.antspam.net`
- Se NXDOMAIN ("no such domain") **non** è uno spammer.
- In caso contrario, la query viene restituita:
 - Un indirizzo 127.0.0.X (dove X è un codice che indica il motivo della lista nera);
 - Un record TXT con ulteriori informazioni.
- RFC-5782 "Blacklist e whitelist DNS".

Poiché gli spammer cambiano MTA abbastanza frequentemente, forse queste liste sono spesso in ritardo nel rilevare gli MTA di spamming, quindi un'altra strategia è quella di non guardare all'indirizzo dell'MTA ma al **contenuto del messaggio**. Se il messaggio contiene un URI (come nel caso del phishing), viene eseguita una ricerca sulla reputazione di quell'URI utilizzando **URI DNSBL** (*dati di reputazione URI*). Molti honeypot/spamtrap sono posizionati per catturare lo spam e classificare l'URI trovato nei messaggi.

Esistono diverse liste (gratuite/commerciali, anonime o meno), come ad esempio:

- *MAPS RBL (Lista buchi neri in tempo reale)*
- *Spamhaus SBL (Spamhaus Block List)*
- *SORBS (Sistema di blocco dello spam e delle reti aperte)*
- *APEWS (Anonymous Postmaster Early Warning System, sistema anonimo di allarme postale)*

Le liste sono gestite da persone non facilmente contattabili, quindi gli URI non sono facili da rimuovere una volta inseriti: si consiglia vivamente di configurare correttamente l'MTA. Attivare/utilizzare l'indirizzo `abuse@domain`, come richiesto da RFC-2142.

Greylisting

Il **greylisting** si basa sull'idea che gli spammer hanno poco tempo a disposizione, perché inviano lo stesso messaggio a milioni di utenti. Gli spammer vorrebbero inviare i messaggi il prima possibile, quindi il greylisting significa che quando qualcuno viene contattato dall'MTA, viene dato un *errore temporaneo* ("prova più tardi") e l'indirizzo dell'MTA viene tenuto in memoria. Se lo stesso MTA ritorna entro un certo intervallo di tempo (ad esempio, 5 minuti), la connessione viene accettata. Questa soluzione si basa sul fatto che in genere uno spammer non vuole perdere tempo e salta la vittima se riceve un errore. Questa soluzione ritarda anche le e-mail valide e il server è più sollecitato, poiché deve mantenere una cronologia dei contatti.

Se non è possibile avere questo tipo di carico sul server di posta in arrivo, è possibile eseguire il **Nolisting** (*greylisting dei poveri*). Si basa sullo stesso presupposto del greylisting e consiste nel definire più di uno scambiatore di posta (nodo progettato all'interno del DNS come punto di contatto per ricevere la posta) per il dominio, assegnando loro una priorità diversa. In questo caso la strategia consiste nel definire almeno 3 mail exchanger: il primo, quello con la priorità più veloce, non risponde; il secondo è quello corretto; il terzo non risponde. Ciò che accade è che gli spammer tentano tipicamente il primo, pensando che sia il più veloce, o l'ultimo, pensando che sia il backup del backup e che non sia ben protetto. Come nella soluzione precedente, anche in questo caso l'ham è in ritardo, perché a partire dal primo l'ham contatterà i diversi server e potrebbe non ottenere presto la risposta.

DKIM (DomainKeys Identified Mail)

Si tratta di una strategia in cui il mittente utilizza una firma per garantire:

- L'identità del mittente;
- L'integrità (parziale) del messaggio.

Ciò avviene tramite una **firma digitale**, creata dall'MSA o dall'MTA in uscita. Questa firma copre alcune intestazioni e parte del corpo ed è verificabile tramite una chiave pubblica (tipicamente inserita nel DNS stesso). Ricordando l'esempio del server di posta di Polito, significa che ogni mail inviata dal mail relay di Polito potrebbe contenere una firma con cui l'MTA di Polito dice "sì, è una mail inviata da me e ho autenticato l'utente" ed è attendibile. Questo sistema è sempre più utilizzato (ad esempio, Gmail, Yahoo).

La firma permette di scartare i messaggi con mittente falso e quindi supporta l'anti-spam e l'anti-phishing.

Un'altra soluzione è l'**SPF (Sender Policy Framework) - RFC 4408** che è una tecnica in cui un dominio di posta elettronica **dichiara quali sono i suoi MTA in uscita**, tramite un record specifico nel DNS.

```
$ 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"
```

Eseguendo un NSLOOKUP su polito.it la risposta è "v=spf1 ptr ~all" che significa che *polito.it* utilizza la versione 1 di SPF, qualsiasi nodo all'interno di Polito che abbia un indirizzo inverso valido (per cui esiste il record PTR) è un mittente valido di posta elettronica e "~all" significa che tutto dovrebbe essere accettato.

Al contrario, per *gmail.com* la stessa query restituisce "v=spf1 redirect=_spf.google.com" e dice che l'elenco esatto è fornito attraverso un redirect e che bisogna cercare nel DNS per "*_spf.google.com*". Quando si esegue la seconda interrogazione, viene fornita un'altra risposta, ovvero l'elenco di tutti i nodi del mondo che sono server di posta in uscita per Google. Una mail proveniente da un utente di Google potrebbe provenire da uno qualsiasi di questi indirizzi; se non è uno di questi indirizzi, è possibile rifiutare i messaggi.

ESMTP

L'**SMTP esteso** è stato definito in RFC-1869 e successivamente incorporato (con SMTP) in RFC-2821. Il protocollo di base e il canale di comunicazione sono gli stessi. I client ESMTP devono **identificarsi** alle parti comunicanti con:

- **Nome host EHLO**

Se il server ricevente parla ESMTP, deve **dichiarare le estensioni** che supporta, una per riga, nella sua risposta a EHLO. Esistono diverse estensioni definite in RFC-4954: quella autenticata aggiunge il comando **AUTH** più le opzioni di MAIL FROM. Sono **necessarie per autenticare un client prima di accettare messaggi da esso**. È utile contro lo spamming perché dopo il comando EHLO il server invia i meccanismi di

autenticazione supportati, il client ne sceglie uno e il protocollo di autenticazione viene eseguito. Se l'autenticazione fallisce, il canale di comunicazione viene chiuso.

Esempio di AUTH negativo

```
220 example.polito.it - SMTP service ready
EHLO mailer.x.com
250-example.polito.it
250 AUTH LOGIN CRAM-MD5 DIGEST-MD5
AUTH PLAIN
504 Unrecognized authentication type
```

login, CRAM-MD5 o DIGEST-MD5. Il client risponde con il metodo *PLAIN*, che non è supportato dal server e quindi il client riceve un errore dal server.

AUTH: metodo LOGIN

```
220 example.polito.it - SMTP service ready
EHLO mailer.x.com
250-example.polito.it
250 AUTH LOGIN CRAM-MD5 DIGEST-MD5
AUTH LOGIN
334 VXNlcmlhbWU6 -----> Username: lioy
bG1veQ== -----> 
334 UGFzc3dvcnQ6 -----> Password: antonio
YW50b25pbw== ----->
235 authenticated
```

Nel caso proposto, il mailer non conosce (o non accetta) il metodo di autenticazione proposto dal client. In nero sono riportati i saluti del server, mentre in blu i comandi del client. Il client dice che sta parlando con il protocollo esteso (EHLO) e che il nome è *mailer.x.com* e il server risponde con la sua identità. Poi il server comunica le tre possibili autenticazioni:

Anche in questo caso, il server fornisce l'elenco dei metodi di autenticazione disponibili e in questo caso il client utilizza il metodo *LOGIN* (che è valido). Il codice 3XX è un intermedio positivo che rappresenta una domanda. Il client risponde, poi un'altra domanda e un'altra risposta. Alla fine, il cliente è stato autenticato. Le stringhe sono solo una codifica base64, quindi è possibile decodificarle con openssl. Il risultato della codifica è quello fornito nell'immagine.

AUT: Metodo PLAIN

Il metodo *PLAIN* è descritto nella RFC-2595 ed evita di inviare i parametri di autenticazione in messaggi separati e utilizza solo una riga:

AUTH PLAIN *id_pwdBASE64*

Id_pwd è definito come segue:

[authorize_id] \0 authentication_id \0 pwd

Si tratta di un identificatore di autorizzazione opzionale, seguito da
e poi l'identificatore di autenticazione, che è obbligatorio, di nuovo un carattere nullo e infine la password.

```
220 example.polito.it - SMTP service ready
EHLO mailer.x.com
250-example.polito.it
250 AUTH LOGIN PLAIN
AUTH PLAIN bG1veQBsaW95AGFudG9uaW8=
235 authenticated
-----> lioy \0 lioy \0 antonio
```

AUTH: metodi di sfida-risposta

Esistono altri metodi (più sicuri) perché sia *LOGIN* che *PLAIN* inviano la password in chiaro, che viene solo offuscata con base64, ma non c'è una vera protezione. Al contrario, CRAM e DIGEST MD5 sono metodi di risposta a sfide simmetriche.

CRAM-MD5

- RFC-2195
- Sfida = *base64(nonce)*
- Risposta inviata dal client = *base64(usr SP hmac-md5(pwd, nonce)_{LHEX})*
 - Utente + spazio + hmac-md5 calcolato sul nonce con una chiave uguale alla password e codificata in esadecimale minuscolo.

DIGEST-MD5

- RFC-2831
- Simile a HTTP/1.1 digest-authentication
- Dichiarato **obsoleto** in RFC-6331 (2011) e sostituito con **SCRAM**

AUTH: Metodo CRAM-MD5

```
220 x.polito.it - SMTP service ready
EHLO mailer.x.com
250-x.polito.it
250 AUTH CRAM-MD5 DIGEST-MD5
AUTH CRAM-MD5
334 PDY5LjIwMTIwMTAzMjAxMDU4MDdAeC5wb2xpdG8uaXQ+
bGlvesA1MGUxNjJzDc5NGZjNDNjZmM1Zjk1MzQ1NDI3MjA5Nw==
235 Authentication successful
lioy hmac(antonio,<69.2012010320105807@x.polito.it>)hex
```

Anche in questo caso, il server fornisce i metodi di autenticazione disponibili e l'utente utilizza CRAM-MD5. A questo punto il server invia la sfida, l'utente invia la risposta e la sfida è riportata nell'immagine. Contiene un numero che di solito si riferisce alla data e all'ora.

Analisi di CRAM-MD5

Vantaggi:

- Permette l'autenticazione del client (password);
- Resistente al replay (sfida = numero casuale + timestamp + FullyQualifiedDomainName);
- Resistente allo sniffing (hash non invertibile).

Svantaggi:

- Nessuna autenticazione del server (ma se viene utilizzato tramite TLS potrebbe essere positivo perché fornisce l'autenticazione del server);
- Memorizzazione in chiaro della password, **a meno che non** vengano memorizzati i passaggi intermedi dell'HMAC (ad esempio, $K \oplus opad$ e $K' \oplus ipad$);
- Attacchi di tipo dizionario se una copia del database è disponibile per gli aggressori.
- Possibili attacchi MITM (channel takeover after CRAM): sempre possibili **dopo** l'autenticazione, che è un problema di qualsiasi autenticazione eseguita solo all'apertura del canale. Se non c'è integrità per ogni messaggio, potrebbe esserci un MITM dopo.

Protezione di SMTP con TLS

RFC-2487 fornisce "*SMTP Service Extension for Secure SMTP over TLS*".

STARTTLS = opzione di EHLO e di comando

Se la negoziazione ha successo, lo **stato del protocollo viene resettato** (il che significa che riparte da EHLO e le estensioni supportate possono essere diverse). Dall'altra parte, se il livello di sicurezza negoziato è *insufficiente*:

- Il client invia immediatamente QUIT e chiude la connessione;
- Il server risponde a ogni comando con il codice 554 (rifiutato per scarsa sicurezza), poiché in SMTP la connessione deve essere sempre terminata dal client.

```
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
```

Servizi di sicurezza per i messaggi di posta elettronica

Quando un MTA parla con un altro MTA non è importante avere un canale TLS perché comunque la posta sarà in chiaro sull'MTA stesso (un TLS protegge solo dal MITM). Piuttosto che proteggere il canale (che non ha senso), poiché la posta elettronica è un servizio hop-by-hop, le e-mail devono essere protette da sole. In questo modo il messaggio è protetto indipendentemente dall'MTA su cui viene memorizzato. Le caratteristiche richieste sono:

- **Integrità (senza comunicazione diretta)**
 - Il messaggio non può essere modificato;
- **Autenticazione**
 - Identifica il mittente;
- **Non ripudio**
 - Il mittente non può negare di aver inviato la posta;
- **Riservatezza (opzionale)**
 - I messaggi non sono leggibili né in transito né quando sono archiviati nella casella postale.

Sicurezza della posta elettronica - idee principali

- **Nessuna modifica all'MTA standard**
 - Ciò significa che, poiché l'aggiunta della firma digitale o la crittografia del messaggio daranno luogo a dati binari, il risultato del messaggio protetto deve essere **codificato per evitare problemi** nel passaggio attraverso gateway (ad esempio, Internet-Notes) o MTA non 8BITMIME.
- **Nessuna modifica all'attuale UA**
 - Interfaccia utente scomoda
- **Con la modifica del presente UA**
 - Migliore interfaccia utente
- **Algoritmi simmetrici**
 - Per la crittografia dei messaggi
 - Con *tasto messaggi*
- **Algoritmi asimmetrici**
 - Per criptare e scambiare la chiave simmetrica
 - Per la firma digitale
- Utilizzare **certificati a chiave pubblica** (ad esempio, X.509) per la non ripudiabilità.

Se tutte queste operazioni vengono eseguite correttamente, la sicurezza del messaggio si basa solo sulla sicurezza dell'UA del destinatario e non sulla sicurezza dell'MTA che non è affidabile.

Tipi di messaggi sicuri

- **Firmato in chiaro**
 - *Messaggio in chiaro* (in modo che chiunque possa leggerlo) + *firma digitale* (come allegato o all'interno del messaggio);
 - Solo chi dispone di un MUA sicuro può verificare la firma;
 - Rischioso perché se la parte di testo non è ben codificata, potrebbe subire modifiche durante la trasmissione e il controllo della firma non corrisponderebbe. Ciò ha a che fare con il fatto che utilizza solo 7 bit per la codifica di un carattere, e il più significativo può essere cancellato da qualsiasi MTA: questo causerebbe la mancata corrispondenza della firma anche se il messaggio è "corretto";
- **Firmato**
 - [messaggio + firma digitale] codificati insieme (ad esempio, base64, uuencode)
 - Solo chi dispone di un MUA sicuro (o esegue le operazioni manualmente) può decodificare e verificare la firma.
- **Crittografato/sviluppato**
 - [messaggio criptato + chiavi criptate] codificato
 - Solo chi possiede un MUA sicuro (e le chiavi!) può decifrare il messaggio.

- **Firmato e imbustato**

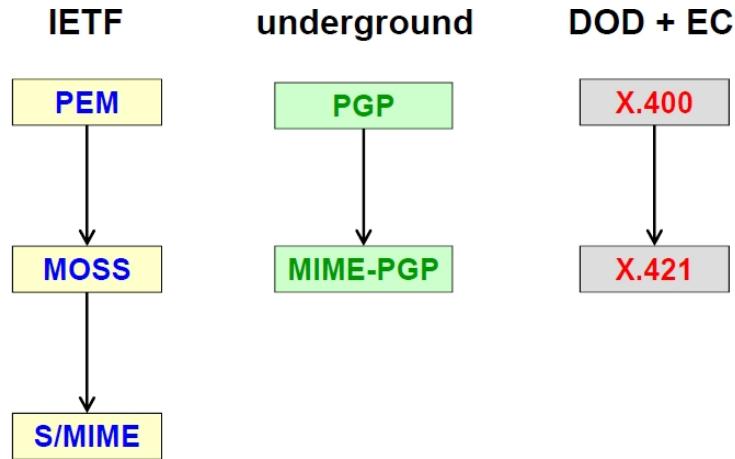
Messaggi sicuri: creazione

- **Trasforma il messaggio in forma canonica**
 - Significa formato standard, indipendente da OS/host/net;
- **MIC (Message Integrity Code)**
 - Fornisce integrità e autenticazione;
 - Tipicamente: messaggio + $(hash(msg)privateSenderKey)$
- **Crittografia**
 - Riservatezza
 - Tipicamente: $(message)MessageKey+(MessageKey)PublicReceiverKey$
 - Si noti che la seconda parte può apparire più volte per più ricevitori.
- **Codifica**
 - Per evitare la modifica da parte dell'MTA
 - Tipicamente: *base64, uuencode, binhex*

Formati di posta elettronica sicuri

Nel mondo di Internet, l'**IETF** ha creato prima lo standard **PEM** che proteggeva solo i messaggi di testo, poi **MOSS** che proteggeva i messaggi basati su MIME e infine l'ultimo standard oggi utilizzato è **S/MIME** (Secure-MIME). Qualcuno non si fida delle CA, quindi ha creato prima lo standard **PGP** e poi **MIME- PGP** per applicarlo ai messaggi MIME ed è tipico del mondo sotterraneo. Il Dipartimento della Difesa e parte della Commissione Europea hanno utilizzato per diversi anni il sistema di posta **X.400**, che era il sistema di posta elettronica più diffuso.

sistema standard per il sistema OSI con un'estensione **X.421** per il multimedia. Questi sistemi non sono più utilizzati e persino i governi utilizzano S/MIME.



PGP (Pretty Good Privacy)

PGP è un sistema di **autenticazione, integrità e riservatezza** per la posta elettronica o i file privati. Ha gli stessi obiettivi del PEM e un'organizzazione simile, ma meno strutturata. È particolare perché non utilizza l'Autorità di certificazione ma il concetto di "**amici fidati e l'algebra di propagazione della fiducia**". È descritto nelle RFC-1991 (informative) e RFC-4880 (OpenPGP). È disponibile per **qualsiasi tipo di piattaforma di sistema** (ad esempio, UNIX, VMS, MS-DOS, Mac, Amiga, ...). L'autore è Phil Zimmerman e il programma è diventato un simbolo della libertà in Internet.

Phil Zimmerman creò il software per conto di alcuni amici e lo rilasciò come freeware nel 1991. Prima del 2000 negli Stati Uniti vigeva una legge che vietava l'esportazione di sistemi di crittografia forti. I sistemi PGP sono apparsi al di fuori degli Stati Uniti, così il governo americano ha deciso di punire Zimmerman e lo ha accusato di esportazione illegale di sistemi di crittografia. Zimmerman viene incarcerato, ma poi la gente inizia a lamentarsi e viene rilasciato su cauzione, ma le indagini e le accuse continuano fino al 1996, quando le accuse vengono ritirate e Zimmerman crea la sua società PGP. Questa PGP Inc. ha avviato una versione commerciale di PGP che in seguito è stata acquisita da NAI. Nell'agosto del 2002 alcune persone esaminarono il codice sorgente di PGP e scoprirono che c'era una backdoor che permetteva a NAI di decriptare tutti i messaggi. Di conseguenza, Zimmerman fu accusato di aver accettato alcuni requisiti per questo tipo di decriptazione. In questo modo, Zimmerman non divenne più un simbolo di libertà per Internet, per cui lasciò NAI e creò una PGP Co. che *non è più open-source e non è disponibile per Linux*.

PGP - certificazione

Poiché non c'è fiducia nella CA, PGP esegue la certificazione nel modo seguente: esiste una chiave pubblica di un utente che viene firmata da tutte le persone che si fidano di quell'utente. Poi la fiducia viene propagata transitivamente, con una certa approssimazione:

- *Completamente*
- *Parzialmente*
- *Non attendibile*
- *Sconosciuto*

Poi, l'algebra della fiducia dice che per fare una fiducia parziale, sono necessari almeno altri tre fidati parziali. Il punto è avere più amici e definire chiaramente il livello di fiducia dell'utente nei loro confronti.

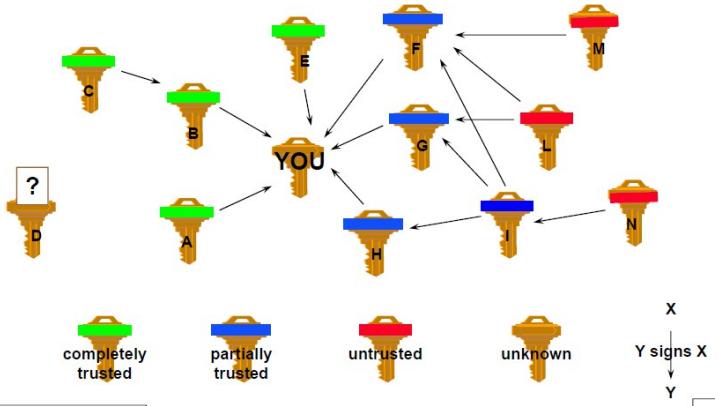
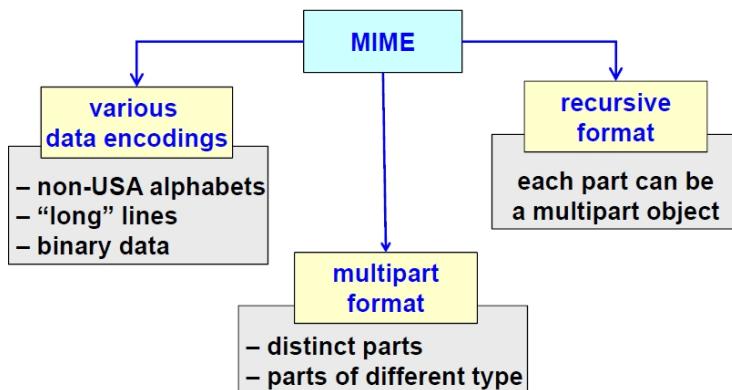


Figura 75 Propagazione della fiducia PGP: nell'esempio "TU" si fida completamente di A, B ed E, quindi C è fidato perché B è fidato. F, G e H sono parzialmente fidati da "TU", quindi I, che è parzialmente fidato da tre pari parzialmente fidati, è parzialmente fidato. Al contrario, L, M e N hanno meno di tre persone di fiducia parziale, quindi non sono attendibili. D è sconosciuto.

PGP - distribuzione delle chiavi

Le chiavi pubbliche sono memorizzate individualmente da ogni utente (nel proprio **portachiavi**). Poi, le chiavi vengono distribuite direttamente dal proprietario (presso una festa PGP) o da un server di chiavi (http, smtp, finger) che è solo un punto di distribuzione in cui viene inserita la chiave con tutte le firme allegate. La fiducia sulla chiave non si basa sul fatto che la chiave viene recuperata dal server di chiavi, ma si basa sul fatto che si deve verificare se tra tutti i firmatari c'è qualcuno di fidato che può dimostrare che la chiave è affidabile. La distribuzione delle chiavi è possibile anche tramite X.500 o DNS.

MIME (Multipurpose Internet Mail Extensions)



MIME è utilizzato per proteggere le e-mail quando contengono anche dati di vario tipo (allegati, immagini, gif, ...). Risolve tre problemi:

1. È possibile utilizzare **diverse codifiche di dati** ed è possibile utilizzare alfabeti non statunitensi, righe "lunghe" (dove "lunghe" significa superiori a 56 caratteri, ovvero la lunghezza massima originale della riga) e dati binari (immagini, filmati, app);
2. MIME è anche un **formato multiparte**

il che significa che un messaggio può contenere diverse parti, ognuna delle quali può avere un tipo diverso;

3. MIME è anche un **formato ricorsivo**, il che significa che qualsiasi parte di MIME può essere a sua volta un oggetto multipart.

Soprattutto le ultime due caratteristiche sono utilizzate nella sicurezza per eseguire la **firma digitale** e la **crittografia**, mentre il supporto di varie codifiche dei dati è importante perché per proteggere un'e-mail con la firma digitale è necessario eseguire lo stesso digest attraverso un algoritmo di hash. Ciò significa che i dati trasmessi non devono essere modificati (ma durante la trasmissione l'MTA ha il diritto di cancellare l'MSB): una codifica appropriata può evitare tale manipolazione.

Posta elettronica multimediale sicura (MOSS o S-MIME)

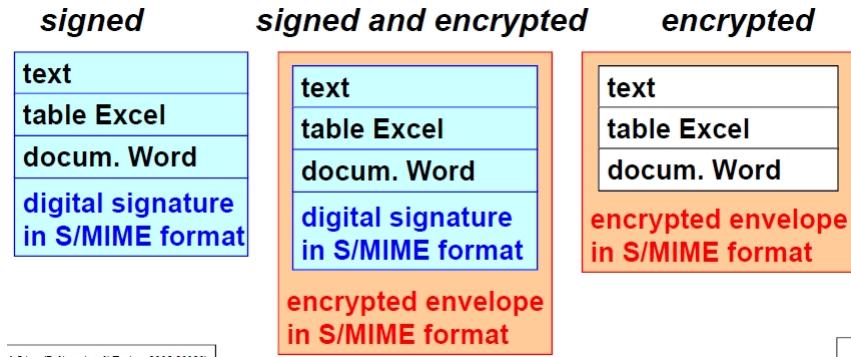


Figura 76 Vari modi in cui un oggetto MIME originale può essere modificato per essere firmato (il primo da sinistra), crittografato (il primo a destra) o sia firmato che crittografato (al centro)

creato: il messaggio da crittografare può essere un messaggio firmato, quindi è possibile che l'oggetto MIME originale sia firmato e crittografato.

RFC-1847

È un'estensione MIME per la sicurezza dei messaggi (standard originale MOSS) per applicare la firma digitale.

Content-Type: multipart/signed;
protocol="TYPE/STYPE";
micalg="...";
boundary="..."

Questo richiedeva di aggiungere un nuovo **Content-Type** al messaggio che era *multipart/signed* con le specifiche del **protocollo** utilizzato (protocollo è usato impropriamente perché non è un protocollo di rete, ma è il tipo di firma digitale che viene applicata al file)

e poi il **micalg** (algoritmo usato per calcolare il codice di integrità del messaggio) e come al solito il boundary, che è la stringa che separa due parti del MIME.

Quando si applica la codifica descritta in RFC-1847, ci sono N parti del corpo, di cui le prime **N-1** sono quelle da proteggere (ad esempio, *content-type: image*) e l'ultima **contiene la firma digitale** (*content-type: TYPE/STYPE*).

S/MIME

A partire dalla definizione originale di MOSS (RFC-1847), è stato definito S/MIME e per qualche tempo MOSS e S/MIME sono stati concorrenti (perché MOSS era stato originariamente definito dall'ITF mentre S/MIME era un prodotto di RSA, una società privata). S/MIME v1 è stato utilizzato solo da RSA, mentre S/MIME v2 è stato pubblicato come serie di RFC informative:

- RFC-2311 "Specifica del messaggio S/MIME v2"
- RFC-2312 "Gestione dei certificati S/MIME v2"
- RFC-2313 "PKCS-1: Crittografia RSA v.1-5"
- RFC-2314 "PKCS-10: sintassi della richiesta di certificazione v.1-5"
- RFC-2315 "PKCS-7: sintassi dei messaggi crittografici v. 1-5".

Nota: il fatto che qualcosa sia pubblicato come RFC non significa che sia stato accettato o approvato dalla comunità Internet.

S/MIME v3, v4

S/MIME si è evoluto e le versioni v3 e v4 sono state create insieme alla comunità Internet. Non si tratta più di una serie di RFC informative, ma i documenti S/MIME v3 e v4 sono *standard proposti*, il che significa che sono approvati dalla comunità Internet.

Entrambi gli standard prevedono la firma/crittografia digitale con certificati X.509 per proteggere i messaggi MIME. Per creare un messaggio di posta firmato, viene aggiunta una parte finale (che deve essere in ultima posizione) calcolando la codifica della firma digitale calcolata su tutte le parti precedenti all'interno del messaggio. Per crittografare il messaggio, il messaggio MIME originale viene considerato come un singolo oggetto e, grazie alla proprietà ricorsiva di MIME, viene creato un oggetto crittografato.

S/MIME v3 (giu '99) poi v3.1 (lug '04) e v3.2 (gen '10)

- *RFC-2633, "Specifiche dei messaggi S/MIME v3".*

- *RFC-2632, "Gestione dei certificati S/MIME v3".*
- *RFC-2634, "Servizi di sicurezza migliorati per S/MIME".*

S/MIME v4 (aprile '19)

- *RFC-8551, "Specifiche dei messaggi S/MIME v4".*
- *RFC-8550, "Gestione dei certificati S/MIME v4".*

Architettura S/MIME

Dal punto di vista architettonico è necessario un formato per rappresentare il corpo protetto del messaggio. È stato:

- **PKCS-7** (su S/MIME v2);
- **CMS** (su S/MIME v3): Sintassi crittografica dei messaggi. È l'evoluzione del PKCS-7 all'interno dell'ITF;
- PKCS e CMS specificano entrambi le caratteristiche crittografiche e i tipi di messaggio (equivalenti a PEM);
- **PKCS-10**: per eseguire una richiesta di certificato attraverso un messaggio di posta elettronica;
- **X.509**: formato dei certificati a chiave pubblica.

S/MIME v4.0 - algoritmi

Firma digitale:

- (DEVE) ECDSA con curva P-256 e SHA-256
- (OBBLIGATORIO) **EdDSA** con curva 25519
- (SCONSIGLIATO) RSA con SHA-256
- (DOVREBBE) RSASSA-PSS con SHA-256

Scambio di chiavi:

- (DEVE) **ECDH** con curva P-256
- (DEVE) ECDH con curva X25519 con HKDF-256
- (DEVE ESSERE SUPPORTATA MA SCONSIGLIATA) Crittografia RSA: utilizzata solo per le applicazioni legacy.
- (DOVREBBE+) RSAES-OAEP

Riservatezza:

- (DEVE) **AES-128-GCM** (riservatezza, autenticazione e integrità) e AES-256-GCM;
- (DEVE ESSERE SUPPORTATO MA SCONSIGLIATO) AES-128-CBC: sarà abbandonato in futuro;
- (DOVERE+) ChaCha20-Poly1305.

Micalg (dipende anche dalla firma digitale):

- SHA-256;
- SHA-512.

S/MIME v3.2 - algoritmi

Firma digitale:

- (DEVE) RSA con SHA-256
- (DOVERE+) DSA con SHA-256
- (DOVREBBE+) RSASSA-PSS con SHA-256
- (DEVE ESSERE SUPPORTATO MA SCONSIGLIATO) RSA con SHA-1
- (DEVE ESSERE SUPPORTATO MA SCONSIGLIATO) DSA con SHA-1
- (DEVE ESSERE SUPPORTATO MA SCONSIGLIATO) RSA con MD5

Scambio di chiavi:

- (DEVE) Crittografia RSA
- (DOVREBBE+) RSAES-OAEP
- (DA SOSTENERE MA SCONSIGLIATO) DHE

Privacy:

- (DEVE) AES-128 CBC
- (DOVREBBE+) AES-192 CBC e AES-256 CBC
- (DEVE ESSERE SUPPORTATO MA È SCONSIGLIATO) DES EDE3 CBC

Micalg (dipende anche dalla firma digitale):

- MD5
- SHA-1
- SHA-224, SHA-256, SHA-384, SHA-512

Tipo MIME

S/MIME crea nuove parti nei messaggi multipart MIME. Queste parti hanno tipi MIME specifici; esempi di tipi MIME sono:

- **application/pkcs7-mime**, viene utilizzato quando:
 - La parte contiene un messaggio **crittografato** (*envelopedData*);
 - La parte contiene un messaggio firmato (*signedData*) indirizzato solo agli utenti S/MIME perché codificato in base64;
 - Messaggio che contiene solo una chiave pubblica (= certificato, in un *corpo signedData degenerato*)
 - Estensione standard: .p7m
 - Sempre codificato in base64, per evitare qualsiasi manipolazione durante il trasporto;
- **Multiparto/firmato:**
 - Messaggi firmati indirizzati anche a utenti che non supportano S/MIME;
 - Il messaggio è chiaro;
 - L'ultima parte MIME è la firma (per RFC-1847) ed è codificata in base64;
 - Estensione standard per la firma: .p7s
- **Applicazione/pkcs10:**
 - Utilizzato per inviare una richiesta di certificazione;
 - Codificato in base64

```
Content-Type: multipart/signed;
protocol="application/pkcs7-signature";
micalg=sha1;
boundary="-----aaaaa"

-----aaaaa
Content-Type: text/plain
Content-Transfer-Encoding: 7bit

Hello!
-----aaaaa
Content-Type: application/pkcs7-signature
Content-Transfer-Encoding: base64

MIIN2QasDDSdwe/625dBxgdhdsf76rHfrJe65a4f
fvVSW2Q1eD+SfDs543Sdwe6+25dBxfdER0eDsrs5
```

Figura 77 Esempio di messaggio che utilizza il tipo di contenuto S/MIME multipart/signed.

Esempi di S/MIME

- **Crittografato:**
 - B64(P7_sviluppato(msg))
- **Firmato (solo per gli utenti S/MIME):**
 - B64(P7_signed(msg));
- **Firmato (per gli utenti generici):**
 - MIME(msg)+B64(P7_signed_detached(msg));
- **Firmato e crittografato:**
 - B64(P7_sviluppato(P7_segnato(msg));
 - B64(P7_signed(P7_enveloped(msg))): questa è l'opzione preferita perché è più robusta agli attacchi DoS. Infatti, se il contenuto è grande, è preferibile verificare l'integrità del messaggio il prima possibile, in particolare prima di decifrarlo, poiché la decifrazione è relativamente costosa.

Nota: *msg* nella notazione precedente è il corpo del messaggio RFC-822.

Nomi in S/MIME

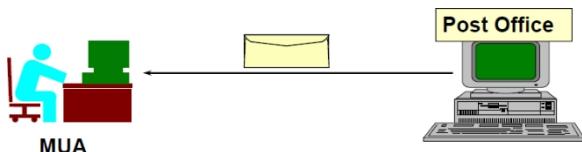
Utilizzato per:

- Selezione del certificato;
- Verifica dell'indirizzo del mittente: l'indirizzo **From** della posta elettronica deve coincidere con l'indirizzo specificato nel certificato;
- S/MIMEv2 utilizza i campi **Email=** o **E=** nel DN del certificato X.509, ma è possibile utilizzare l'estensione **subjectAltName** con codifica rfc822.

S/MIMEv3 richiede l'uso dell'estensione **subjectAltName** con codifica rfc822: ciò significa che il DN nel certificato non deve contenere l'indirizzo e-mail, ma deve essere presente nell'estensione **subjectAltName**, codificata secondo rfc822.

Servizi di posta elettronica client-server

Finora è stata affrontata la protezione del messaggio di posta in sé: con SMTP è possibile proteggere la trasmissione in uscita, ovvero un'operazione di push sull'MSA o sull'MTA. Ricordando lo schema generale della Figura 74, l'operazione di pull corrisponde alla verifica della presenza di e-mail all'ufficio postale. Per eseguire questa operazione, le proprietà di sicurezza necessarie sono:



- **autenticazione dell'utente**: il server di posta deve essere sicuro di consegnare le e-mail solo all'utente a cui appartengono;
- **autenticazione del server**: il client deve essere sicuro di non ricevere e-mail false da un server falso;
- **riservatezza/integrità dei messaggi di posta**:

 - **sul server**: per ottenere ciò, questo tipo di protezione deve essere affrontato dal mittente, perché il destinatario non ha alcun controllo su ciò che viene memorizzato nell'ufficio postale;
 - **durante il trasporto**.

Quindi, se il messaggio richiede una protezione, il mittente deve applicare S/MIME, e in questo modo il messaggio è protetto sia sui server (l'ufficio postale stesso o anche gli MTA intermedi) che in transito.

POP (Protocollo Post-Office)

POP ha due versioni principali:

- **POP-2 (RFC-937)**: non più utilizzato, in quanto non prevedeva nemmeno l'autenticazione tramite password, ma solo la dichiarazione del nome utente;
- **POP-3 (RFC-1939)**: non è più utilizzato, poiché prevede l'autenticazione dell'utente tramite una password in chiaro;
- Autenticazione dell'utente **APOP (solo)**, mediante sfida:

- Il comando APOP sostituisce l'insieme dei comandi USER + PASS
- La sfida è la parte della riga hello contenuta tra le parentesi < ... > (comprese le parentesi)
- Sintassi:
 - Risposta dell'utente APOP alla sfida
- Risposta = MD5(challenge+password), codificato in esadecimale;

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
```

- Supportato da Eudora MUA.

Figura 78 Esempio di utilizzo del protocollo POP3

Figura 79 Esempio di utilizzo del protocollo 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
```

- **K-POP** è un sistema di autenticazione reciproca tramite ticket, ma richiede l'uso di Kerberos.

IMAP (Internet Mail Access Protocol)

L'autenticazione predefinita dell'utente avviene tramite nome utente e password inviati in chiaro: tuttavia, supporta OTP, Kerberos o GSS-API.

RFC-2595 "Utilizzo di TLS con IMAP, POP3 e ACAP"

Per risolvere il problema dell'autenticazione del server e quello dei protocolli che utilizzano una trasmissione non sicura di nome utente e password, RFC-2595 documenta l'uso di TLS con POP3 e IMAP.

È previsto che prima si apra il canale di comunicazione, poi si negozino le caratteristiche di sicurezza tramite un comando dedicato (in modo simile a quanto avviene con HTTP e SSL/TLS):

- **STARTTLS** per IMAP e ACAMP;
- **STLS** per POP3.

Il client e il server devono essere configurati per rifiutare nome utente e password se inviati in chiaro. In questo modo il client confronta l'identità del certificato con quella del server.

Porte separate per SSL/TLS?

L'uso dei comandi precedenti per trasformare un canale normale in uno sicuro è oggetto di dibattito nella comunità della sicurezza: l'alternativa suggerita è quella di utilizzare porte diverse per la versione sicura del protocollo. Ad esempio, nel caso di http, è possibile utilizzare la comune porta 80 e poi usare il comando STARTTLS per rendere sicura la comunicazione, oppure è possibile attivare prima TSL sulla porta TCP 443 e poi eseguire http su di essa. Attualmente, l'uso di porte diverse è sconsigliato dall'IETF per i seguenti motivi:

- Coinvolge URL diversi (ad esempio, http e https);
- Coinvolgere un modello sicuro/insicuro non corretto (ad esempio, SSL a 40 bit è sicuro? È insicura un'applicazione senza TLS ma con SASL?);
- Non è facile implementare "usa TLS se disponibile";
- Raddoppia il numero di porte necessarie.

Tuttavia, l'utilizzo di porte diverse presenta alcuni

vantaggi:

- **È semplice filtrare il traffico sui firewall a filtro di pacchetti;**
- **TLS con autenticazione del client consente di non esporre le applicazioni ad attacchi:** ciò è dovuto al fatto che se il canale TLS viene aperto prima della connessione al server delle applicazioni è possibile ridurre al minimo la superficie esposta a un attaccante, la cui autenticazione fallirebbe. Questo è difficile da implementare se il TLS viene attivato dopo, perché a quel punto l'attaccante sarebbe già connesso al server dell'applicazione (vedere Autenticazione nelle applicazioni web).