

SICUREZZA SISTEMI INFORMATIVI

0) INTRODUZIONE

SICUREZZA INFORMATICA è l'insieme di prodotti, servizi e regole organizzative che proteggono i sistemi informatici aziendali da accessi indesiderati, per garantire la riservatezza delle informazioni e assicurare il corretto funzionamento dei servizi (anche a fronte di eventi imprevedibili).

Conseguenze economiche relative all'errata sicurezza informatica sono perdita di soldi, costi di ripristino del servizio, perdita di produzione e la perdita di affidabilità/reputazione aziendale.

La tecnologia di **ATTACCHI INFORMATICI** si sta sviluppando rapidamente e open-source, mentre la **DIFESA** è **reazionaria** (non previene, ma cura); molti sistemi connessi ad Internet hanno una sicurezza debole. I programmati spesso non fanno "**secure programming**" (usano funzioni non sicure, non sapendole) e i cyber-criminali non vengono processati a causa di motivi geografici/internazionali.

Gli **attacchi** sono "**Business Email Compromise**" (e-mail aziendali hackerate), "**Computer Data Breach**", "**Ransomware**" (malware che cifra tutti i dati dell'hard-disk e viene richiesto un riscatto per avere la chiave di decifratura) ... Gli **attaccanti** sono il crimine organizzato, gli stati, ma anche utenti interni.

"ACCESS VECTORS" (ovvero i punti di partenza degli attacchi) sono "Exploitation of vulnerabilities", "Phishing" e "Credential abuse". I server (WebApp e e-mail) sono gli asset più interessati, ma anche dispositivi (desktop, portatili e smartphone).

Continua l'aumento degli attacchi basati su sfruttamento delle **vulnerabilità dei sistemi** (codice con dei bug al suo interno) per **ransomware** e estorsioni, ma anche gli attacchi tramite la **Supply Chain** (attacco dei **fornitori**/partner aziendali, che a differenza dell'azienda principale hanno una sicurezza minore e quindi tramite i fornitori accedo all'azienda principale tramite il canale privilegiato del fornitore).

Alla base ci sono **problemi tecnologici** (reti insicure che lavorano in chiaro o in broadcast, linee di comunicazioni condivise e router di terze parti, autenticazione debole degli utenti, software con bug) e problemi di programmazione/progettazione/configurazione. Con il progresso tecnologico, è **aumentata la complessità del sistema ed è quindi più difficile verificarne la correttezza** (quindi contro il principio **KISS** [kiss it simple stupid]).

Vediamo alcuni degli **ATTACCHI BASE**:

- **PACKET SNIFFING** = lettura dei pacchetti destinati ad un altro nodo di rete (facile in reti broadcast), permettendo di intercettare password, dati etc...
- **IP SPOOFING** = falsificazione dell'indirizzo di rete del mittente (falsificazioni di dati e accessi non autorizzati) [IP solitamente, ma anche MAC si può fare]
- **SHADOW/FAKE SERVER** = elaboratore che si pone come un fornitore di un servizio senza averne il diritto (si fa *sniffing* della richiesta e *spoofing* della risposta usando un server ombra più veloce di quello reale o bloccando il server reale con attacchi DDoS) [risolto con autenticazione del server]
- **HIJACKING/MITM (Man-In-The-Middle)** = *data spoofing*: si prende il controllo di un canale di comunicazione e si manipolano i pacchetti scambiati (tramite MITM logico o fisico)

- **DoS (Denial-Of-Service)** = tenere impegnato un host per impedirgli di fornire i suoi servizi (saturazione di posta, log, ping...) [no soluzioni, solo metodi per gestire più traffico]; se viene fatto su molti nodi, **DDoS** che crea una **Botnet** (molti nodi chiamati *daemon*, zombie o *malbot* che vengono controllati da remoto da un master)
- **MALWARE**:
 - **Virus** = provoca danni e si replica, propagati dagli umani
 - **Worm** = provoca danni perché si replica (satura), propagazione automatica
 - **Trojan** = vettore di malware (strumento usato per portare malware)
 - **Backdoor** = punto di accesso non autorizzato
 - **Rootkit** = strumenti per accesso privilegiato (nascosti, installati per prendere i privilegi di admin del sistema)
- **RANSOMWARE** = tipologia di MALWARE che usa la crittografia per cifrare la memoria/memorie degli utenti, e poi chiedono un riscatto per decifrare i dati (una minaccia sempre più diffusa negli ultimi anni)

INDUSTRIA del CYBERCRIME → **CaaS (Crime-as-a-Service)** = attività criminali ben strutturate, con servizi criminali offerti a buon prezzo, che si appoggiano su server messi a disposizione (ad hoc) o compromessi. Posso comprare componenti di cui necessito e fare le cose io, oppure appoggiarmi a loro ed usufruire dei suoi servizi (**prima servivano molte risorse, quindi attacchi studiati e grossi**, ora invece costa poco, quindi **attacchi a tappeto per provare con tutti**) [**servizi offerti: phishing, exploit e zero-days, malware, call center, DDoS e money mule**]

Gli **errori umani** sono ritenuti l'elemento più debole nella catena della cybersecurity (quindi problemi hardware [farsi rubare l'hw] ma anche problemi "non-tecnologici" [scarsa awareness, troppa fiducia, che può portare infatti anche alla partecipazione inconsapevole dell'utente all'attacco che viene manipolato*]); è fondamentale:

- usare password forti e obbligare a cambiare quelle di default
- applicare subito le patch di sicurezza per limitare le esposizioni agli attacchi
- analizzare i componenti che si danno ai propri clienti

Alcuni esempi di attacchi legati al **SOCIAL ENGINEERING** (1):

- **PHISHING** = attirare (tramite mail, msg, social...) su un sito falso, rubare credenziali di accesso, far installare involontariamente un malware; varianti specializzate sono:
 - **spear phishing** → include dati precisi/personali per aumentare la credibilità del messaggio
 - **whaling** → mirato a persone importanti (CEO, CTO...)
- **PRETEXTING** = metodo per convincere le persone a svolgere qualche attività (costruendo storie e usando informazioni precise/personali)

Alcuni **eempi storici avvenuti**:

- **Stuxnet** = nuovo attacco molto sofisticato che ha danneggiato fisicamente sistemi di centrifughe per l'arricchimento dell'uranio facendoli girare molto velocemente; si propaga anche ad altri sistemi (1° esempio di **malware per sistemi cyberfisici**, che usò worm + virus per windows e alcune vulnerabilità note e zero-days, ma anche una chiavetta corrotta inserita per errore)
- **APT (Advanced Persistent Threat** → minacce persistenti avanzate che hanno controllo remoto e continuo su sistemi compromessi) su caso Black Energy (BE) = ha generato blackout, staccando l'elettricità a molte persone
- **IoT** caso Mirai = è un cyberworm che aveva preso il controllo di molti nodi IoT (che diventano zombie parte di una botnet per attacchi DDoS), inizialmente molti baby-monitor (avevano preso anche controllo della videocamera) [è stato poi rilasciato in open-source]

- **BlueBorne** = usò connessione Bluetooth per controllare i veicoli a distanza, diffuso via wireless (car hacking)

SICUREZZA

- **Proprietà:**
 - **Autenticazione** (semplice/mutua) [authentication] =
 - **Autenticazione della controparte** [peer authentication] = qualcuno chiede, qualcuno verifica
 - **Autenticazione dei dati** [data/origin authentication] = conferma chi ha generato i dati
 - **Autorizzazione/controllo accessi** [authorization] = garantire l'accesso a determinate risorse
 - **Integrità** [integrity] = no modifica, cancellazione/filtraggio di comunicazioni (es. attacchi replay che re-inviano la comunicazione più volte)
 - **Riservatezza** [privacy, confidentiality] = essere mantenuto segreto; solo chi è autorizzato può fare delle cose
 - **Non Ripudio** [non repudiation] = prova formale (usabile in tribunale) che dimostra in modo innegabile l'autore dei dati
 - **Disponibilità** [availability]
 - **Tracciabilità** [traceability]
 - **Protezione dei dati** = per ciascuna proprietà considerare sempre 2 casi:
 - *data in transit* → quando i dati sono trasmessi su un canale di comunicazione
 - *data at rest* → quando i dati sono memorizzati su un dispositivo
- **Principi:**
 - **Security by design** = fin dall'inizio progettato per essere sicuro, per coprire tutti i rischi possibili
 - **Security by default** = progettato per avere la migliore security, senza nemmeno far sapere allo user che è così
 - **Defense in depth** = la sicurezza viene applicata integrando persone, tecnologia e operazioni (più livelli di sicurezza dell'organizzazione)
- **Principi del NIST:**
 - **need-to-know** = dobbiamo minimizzare le info che diamo in giro, facendo che ognuno abbia solo quello che gli serve per fare le sue operazioni [target = persone]
 - **least privilege** = dare i privilegi minimi all'utente affinché faccia quello che gli serve fare (ma non di più) [target = processi]
- **3 Pilastri della sicurezza:**
 - 0) Planning
 - 1) Avoidance
 - 2) Detection
 - 3) Investigation

1) GESTIONE dei RISCHI INFORMATICI

GESTIONE DEL RISCHIO

La gestione dei rischi informatici è ormai un'attività essenziale in ogni azienda: quantificare i rischi mediante un processo ripetibile e oggettivo, per poterli ridurre entro limiti "tollerabili".

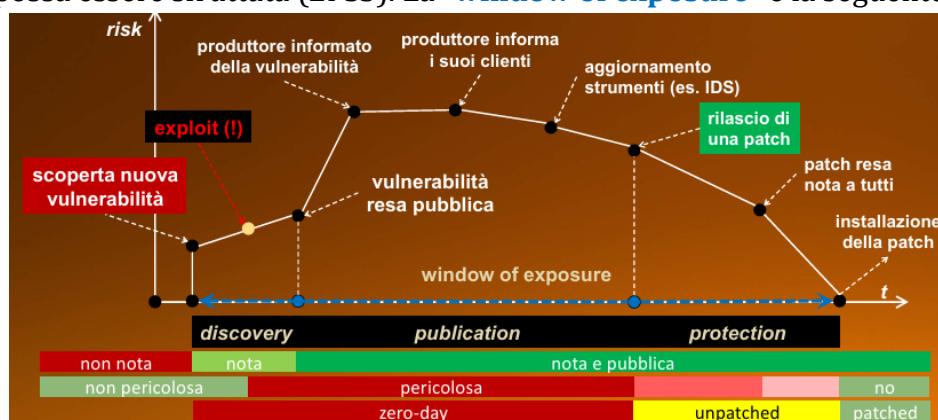
Terminologia:

- **ASSET** = insieme di beni, dati e persone necessarie all'erogazione di un servizio (quindi le risorse)
- **DEBOLEZZA** = predisposizione del sistema a manifestare problemi e vulnerabilità
- **VULNERABILITÀ** = debolezza sfruttabile da attaccanti per violare proprietà di sicurezza
- **MINACCIA** = evento intenzionale/accidentale, che se sfruttato può far perdere proprietà di sicurezza
- **ATTACCO** = minaccia di tipo "evento intenzionale"
- **EVENTO NEGATIVO** = minaccia di tipo "evento accidentale"

CVE (Common Vulnerabilities and Exposures):

- **Vulnerability** = errore nel software che può essere usato per prendere accesso ad un sistema/rete
- **Exposure** = errore nel software/nella configurazione di sistema che non direttamente compromette, ma che potrebbe essere usata per un attacco

Vengono riportate alla **MITRE Corporation** (per classificarle) che contatta l'ente che può subire l'attacco a causa della vulnerabilità [che quindi può sistemare prima del danno] e poi ufficializza la CVE. Le CVE sono identificate da un ID e associate ad un punteggio (*CVSS*; 0-10 in base alla gravità) e ad una probabilità che possa essere sfruttata (*EPSS*). La "**window of exposure**" è la seguente:



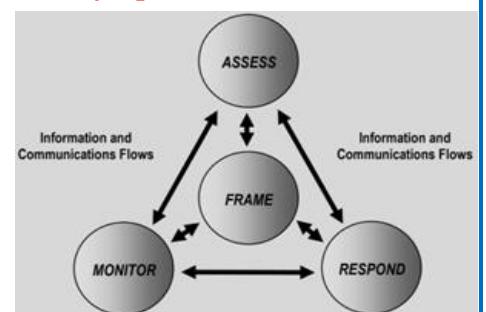
CWE (Common Weakness Enumeration) = tassonomia/classificazione degli errori di programmazione, configurazione, gestione (presenti in hw e sw in produzione) che devono essere monitorate da chi sviluppa e da chi usa l'hw/sw. Sono **debolezze** che potrebbero diventare vulnerabilità (non lo sono già → non è detto che una CWE diventi una CVE). Un esempio è la mancata validazione dell'input (potrebbe permettere una *SQL injection* [chi compila potrebbe scrivere del codice SQL che verrà quindi preso dal DB e manderà tutto in malora]).

Quindi **come facciamo a GESTIRE i rischi informatici in modo scientifico/ripetibile?**

RMF (Risk Management Framework), proposto dal **NIST**, comprende

4 fasi:

- **preparazione (frame)** → output: descrizione accurata del proprio sistema, del processo da adottare per valutare il rischio e strategie per la gestione dei rischi
- **valutazione dei rischi (assess)** → identificazione delle minacce, debolezze, vulnerabilità e conseguenze
output: valutazione di tutti i rischi (o quanti più possibile)



- **identificazione + implementazione** delle mitigazioni ([respond/risk mitigation](#)) → output: soluzioni per la mitigazione/eliminazione dei rischi identificati; valutazione delle soluzioni identificate [\$, t, impatto] → scelta delle soluzioni + loro implementazione
- **aggiornamento costante dell'analisi** ([monitor](#)) → valutare le mitigazioni applicate (+ eventuali rifacimenti del processo RMF)

All'interno della valutazione dei rischi (**risk assessment**), c'è la [sottofase Vulnerability Assessment](#) (valuta l'esposizione aziendale alle vulnerabilità in 4 fasi):

- **planning** = cosa includere nella valutazione (scope) e quando farlo (schedule [è onerosa])
- **information gathering** = acquisire più informazioni possibili sugli obiettivi in scope
- **scanning** = sulla base delle informazioni acquisite, cerca le vulnerabilità note; vari tipi:
 - **host discovery** (cercare le macchine del difensore) → ICMP scan ("ping"), ARP scan, TCP SYN/ACK, UDP scan
 - **port scanning** → classifica le porte come open (in ascolto e raggiungibile), closed (non in ascolto), filtered (non raggiungibile)

Gli scanner provano diverse sequenze di pacchetti/connettoni per identificare le versioni e gli OS delle macchine (così da attaccarle meglio) [oggi si usano anche algoritmi di AI/ML per migliorare l'identificazione ancora di più] [[TCP Sequence Predictability Classification](#) → conoscere il n° di sequenza del prossimo pacchetto TCP per fare spoofing]

- **report dei risultati** = include le proposte per le mitigazioni

Ci sono dei "**vulnerability assessment tool**" per automatizzare questo processo di vulnerability assessment (es. GVM [un tempo "OpenVAS"])

ANALISI e VALUTAZIONE DEL RISCHIO

Definire precisamente i processi aziendali necessari alla gestione del rischio (creare processi, ruoli e responsabilità); uno stimolo per farlo sono la compatibilità con standard e normative. I risultati ottenuti sono anche al di là della sicurezza (classificazione degli asset, identificazione delle minacce e rischi).

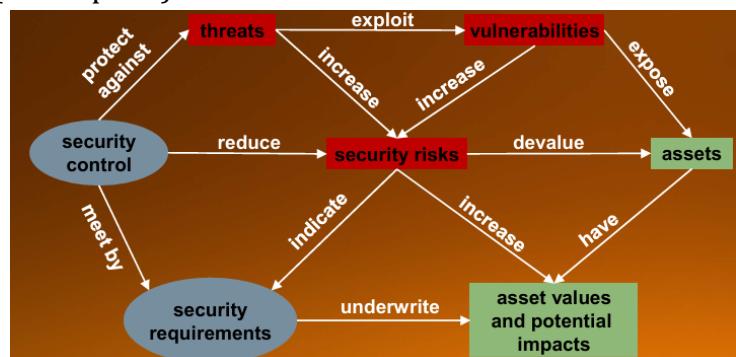
Quindi, una volta che conosco le parti esposte del mio sistema (dalla gestione del rischio), devo essere in grado di dare un **peso** al rischio (non possono affrontarli tutti insieme, devo scegliere da cosa partire):

- **danno** max/medio
- **tempi di recupero** della piena funzionalità
- **probabilità** che accada

Approcci per **dare questo "valore"** di peso al rischio:

- **quantitativo** (numeri)
- **qualitativo** (categorie)
- **semi-quantitativo** (valori numerici indicativi o scale di valori) [il migliore]
- approccio basato sulla "**monetization** (\$ max persi)

Relazioni tra concetti legati al rischio:



Approcci per l'analisi sono:

- **threat-oriented** = in funzione delle minacce
- **asset/impact-oriented** = in funzione degli asset (impatti valutati sulla base della capacità delle minacce di compromettere gli asset)
- **vulnerability-oriented** = in funzione delle vulnerabilità

MITRE ATT&CK contiene tutti gli attacchi (è un *knowledge base* [una sorta di DB] che descrive le tecniche di attacco divise in **macrocategorie** con alcuni esempi per i vari attacchi), mentre **MITRE D3FEND** contiene tutte le difese (è un *knowledge graph* che descrive tutte le contromisure per i vari attacchi prima elencati) [c'è anche **OWASP** che fa queste cose, ma per le applicazioni *Web* (calcola il rischio come likelihood * impact), elencando le top 10 threats]

VULNERABILITY MANAGEMENT → serve una gestione per ordinare i difetti identificati per importanza e per prendere decisioni su dove e come intervenire (fix, patch...), con controlli basati su *actual state vs desired state*, al fine di **automatizzare la gestione delle vulnerabilità**.

Secondo il **NIST**, un'azienda dovrebbe avere un workflow ben definito, con **ruoli** ben precisi che le aziende dovrebbero creare:

- **Software Flaw Manager** (per chi sviluppa SW e deve gestire i processi per scoprire e fixare debolezze e vulnerabilità)
- **Patch Manager** (per chi usa SW e deve gestire le vulnerabilità pubbliche del sw, con scansioni per applicare le patch e verificare che tutto funzioni) [*Patch management is the process for identifying, acquiring, installing, and verifying patches for products and systems*]; le **PATCH** correggono problemi e servono in questo ambito a mitigare l'esposizione di exploitation o ad aggiungere funzionalità. Inoltre, come sempre, se sono tante e grosse vanno applicate in ordine in base alla priorità (ci sono anche *aggregated patch* [più vulnerabilità risolte in 1 patch])

Con quale ordine applicare quindi le patch secondo il NIST?

- **subsets**: applicarle prima a piccoli gruppi
- **standard first**: applicarle prima a sistemi standard che si sa che non avranno grossi problemi
- **difficult ones later** (sistemi multipiattaforma, legacy, non-standard...)
- **cover all** (mobile, computer off-line durante il patching...)

⚠ C

2) CRITTOGRAFIA SIMMETRICA

Una sequenza casuale (**true-random**) ed una pseudo-casuale (**pseudo-random**, ovvero generata mediante un generatore deterministico) sono indistinguibili a livello pratico, ma i generatori di numeri pseudo-casuali possono generare solo un numero finito di sequenze (n° di sequenza = **seed**).

Processo di trasformazione della CRITTOGRAFIA:



⚠️ Un'operazione crittografica deve sembrare casuale (un attaccante non può ricavarne info)

Terminologia:

- **PLAINTEXT** (o **CLEARTEXT**) [P] = messaggio in chiaro
- **CIPHERTEXT** [C] = messaggio cifrato
- Encrypt/Decrypt (o **cipher/decipher**) = cifrare/decifrare

CRITTOGRAFIA SIMMETRICA (o a **chiave segreta**) = chiave unica/comune a mittente e ricevente; basso carico di elaborazione [OK per cifratura di dati]:

$$C = enc(K, P)$$

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

Un esempio storico è il *cifrario di Cesare*; oggi un tool di questo tipo è **OpenSSL** (composto di moduli specializzati; tool di riferimento per le esercitazioni sulla crittografia). Ma come possiamo **condividere in modo sicuro la chiave segreta** tra mittente e ricevente? Questo è infatti il problema di questi algoritmi.

Alcuni **algoritmi di cifratura simmetrica** sono **AES** (algoritmo ufficiale del governo americano), **DES**... Sono caratterizzati dal tipo di trasformazioni, lunghezza delle chiavi (40-256bit), tipologia (blocchi o stream), modalità d'uso (tanti dati o pochi dati), computazione (seriale o parallelo) ...

⚠️ **STO (Security Through Obscurity)** = principio per cui si ritiene sicuro qualcosa solo in base al fatto che è tenuto segreto (un'illusione inutile); infatti il **principio di Kerckhoffs** fa capire la forza della crittografia: **solo le chiavi vanno tenute segrete, gestite da sistemi fidati e devono essere di lunghezza adeguata; è meglio che gli algoritmi di cifratura/decifratura siano resi pubblici** (affinché vengano studiati e nel caso rivelate eventuali debolezze)

Infatti se l'algoritmo di crittografia è stato ben progettato e le chiavi (lunghe **n bit**) sono tenute **segrete**, allora l'unico attacco possibile è il **brute force** (provo fin che non indovino), che richiede:

$$n^{\circ} \text{ tentativi} = T = 2^n$$

La forza degli algoritmi simmetrici dipende dal **tempo necessario a fare tutti i tentativi brute force** (infatti chiave più lunga = più forte [a parità di algoritmo]). Per ogni tentativo bisogna considerare il tempo per la **decifratura** e il tempo per **valutare se il risultato è plausibile**.

Gli **algoritmi invecchiano** (causa tecnologia che avanza, quindi tutti sono destinati a indebolirsi). Si chiama **CRITTOANALISI** fare ricerche e osservare i messaggi scambiati per ridurre lo spazio di ricerca della chiave (magari si capiscono alcuni bit, quindi ci sono meno combinazioni disponibili).

Anche le **chiavi invecchiano** e vanno sostituite (anche se lunghezza è abbastanza, va cambiata per questioni di tempo [tempo per provare combinazioni aumenta] e per questioni legate al suo uso); le chiavi devono essere sempre **random** però.

STREAM o BLOCCHI

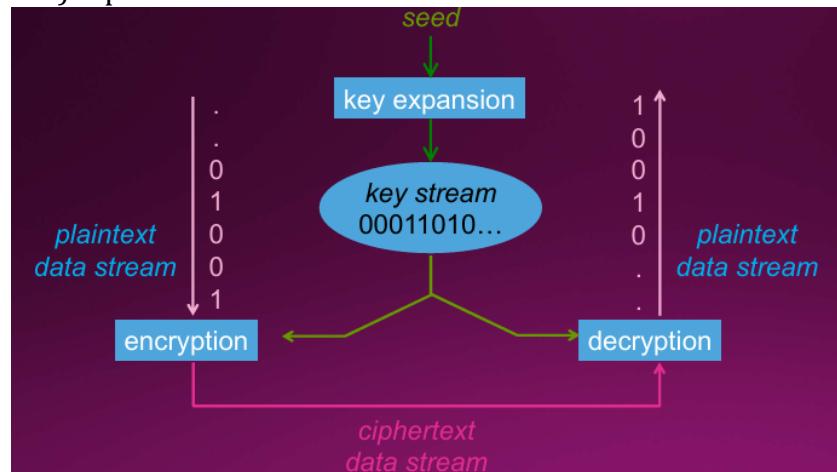
Un operatore di “*confusione*” ideale è lo **XOR** (dove se l’input è casuale, anche l’output sarà casuale ugualmente), in quanto è anche disponibile su ogni CPU.

\oplus	0	1
0	0	1
1	1	0

Gli **algoritmi simmetrici di tipo STREAM** operano su un flusso di dati senza richiederne la divisione in blocchi (tipicamente su 1 bit/byte alla volta).

Algoritmo **ideale**: 1-time-pad → richiede una **chiave lunga tanto quanto il msg da proteggere**

Algoritmi **reali**: usano **generatori di sequenze pseudo-casuali pilotati dalla chiave** (se uso stessa chiave, ottengo stessa sequenza). Operano con **XOR bit-a-bit sui dati**



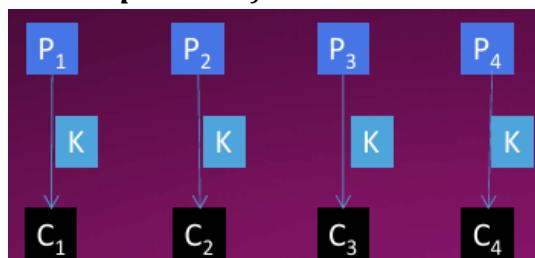
Degli esempi sono **Salsa20** (ora obsoleto; fa “20” volte il mixing sul seed + input) e **ChaCha20** (migliorato: > *diffusione* dei bit, > veloce su alcune architetture; operazione base = **ARX** [Add-Rotate-XOR] su 32 bit). Hanno sostituito **RC4** come algoritmo di stream (il quale non usa **nonce**).

Gli **algoritmi simmetrici a BLOCCHI** trasformano **1 blocco di dati alla volta** (dimensione dei blocchi **fissa** [128bit nei moderni]), con trasformazione pilotata dai bit della chiave (che decidono come e cosa trasformare):

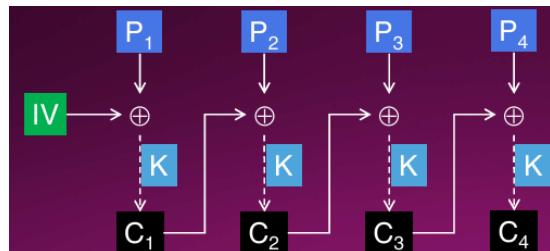


Come si applica un **algoritmo a blocchi** a dati in quantità diversa da quella del blocco base?

- per **cifrare in quantità >**:
- o **ECB** (Electronic Code Book) = $C_i = enc(K, P_i)$; sconsigliato (specialmente su msg lunghi più di 1 blocco) in quanto è possibile scambiare 2 blocchi qualunque e **cifra allo stesso modo blocchi identici** (quindi soggetto ad attacchi **known-plaintext**)



- **CBC** (Cipher Block Chaining) = $C_i = enc(K, P_i \oplus C_{i-1})$; forte perché ogni blocco viene cifrato anche in funzione del blocco precedente che quindi aggiunge "salting"; richiede però random C_0 (ovvero il primo blocco, chiamato **IV** [Initialization Vector])



- per **cifrare in quantità <**:
- **Padding** (allineamento) → dimensione blocco = B, dimensione dati = D (non multiplo di B); aggiungo bit per rendere D + padding multiplo di B
- **cipher stealing** [simulano algoritmo di stream (CFB, OFB)]
- per **cifrare e numerare i pacchetti** → CTR (Counter mode)

INTEGRITÀ'

Un **Man-In-The-Middle (MITM)** che intercetta una comunicazione cifrata non può leggerla, ma può **modificarla** in modo imprevedibile. Il principio alla base dell'integrità è:

- mittente: usa funzione per generare una prova di come sono fatti i dati in un momento in cui è sicuro che siano corretti (prima di spedirli)
- destinatario: ricalcola la stessa funzione sui dati a sua disposizione (appena ricevuti)
- confronto → se i 2 valori calcolati da mittente e destinatario sono uguali, sono integri (altrimenti no) [confronto deve avvenire in modo sicuro]

MESSAGE DIGEST = riassunto a lunghezza fissa del msg da proteggere; deve essere veloce da calcolare, impossibile (o quasi) da invertire e **deve essere difficile per gli attaccanti "generare collisioni"** (collistione = 2 msg diversi con stesso riassunto). Viene spesso usato per non lavorare sul messaggio completo (perché è molto grande); il digest solitamente viene calcolato con una funzione di hash (crittografica).

⚠ Problemi relativi alle collisioni:

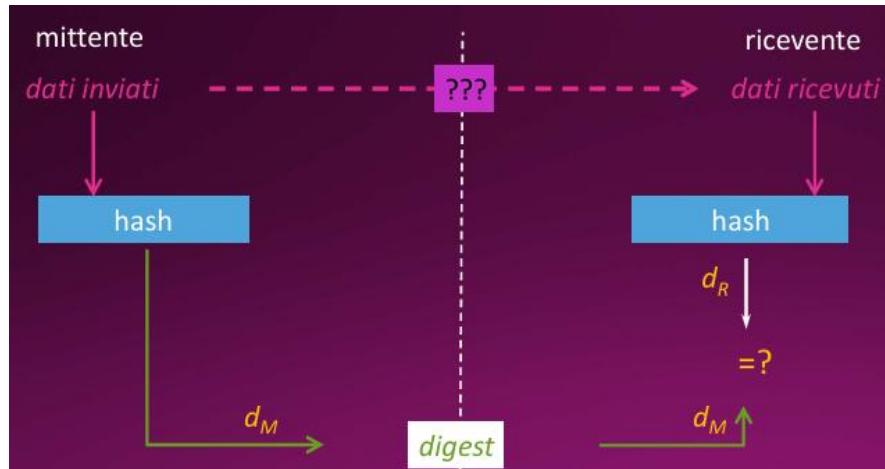
- **Collisione di tipo 1**: dato un messaggio m , creare un messaggio $m' \neq m$ che abbia lo stesso digest di m (**esempio di attacco**: intercetto un documento, ne creo uno con lo stesso digest e lo sostituisco) [funzione di digest è sicura se servono $> 2^{\text{len}(m)}$ tentativi]
- **Collisione di tipo 2**: creare 2 messaggi m e m' con lo stesso digest (**esempio di attacco**: creo 2 msg con stesso digest e li invio a 2 persone diverse che sono convinte di aver ricevuto lo stesso documento; faccio validare il documento, poi uso l'altro) [funzione di digest è sicura se servono $> 2^{\text{len}(m)/2}$ tentativi]

Funzioni di hash sono "sicure" se ha resistenza pari a 80bit (quindi digest di almeno 160bit); ma in realtà gli algoritmi con digest da 160bit sono obsoleti, noi consideriamo sicuri solo algoritmi con digest a partire da 256bit (quindi da SHA-2 [256, 512...] a SHA-3)

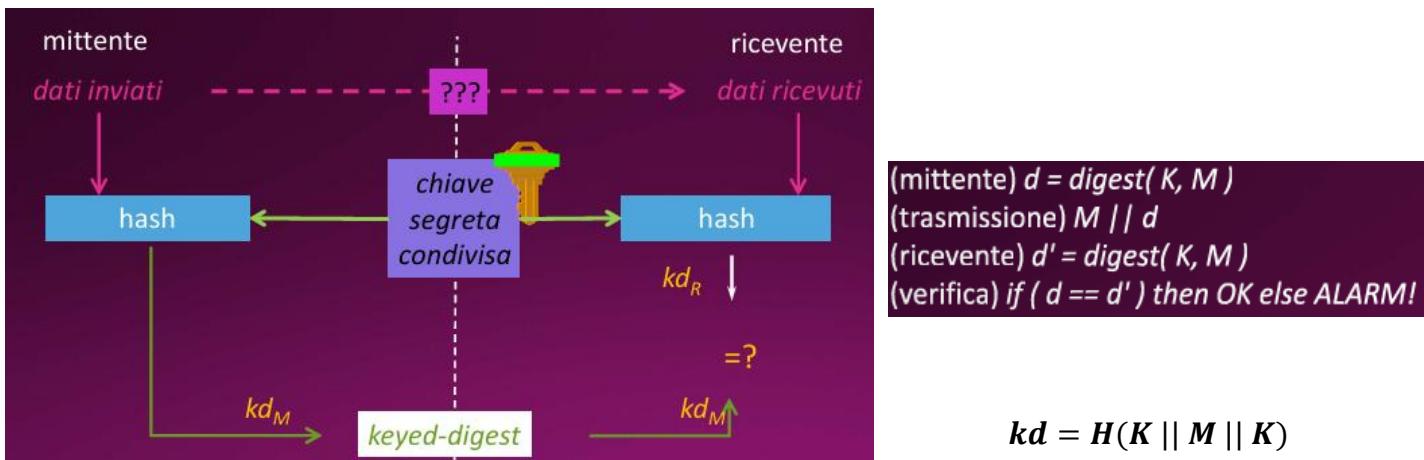
nome	blocco	digest	definizione	note
MD2	8 bit	128 bit	RFC-1319	obsoleto
MD4	512 bit	128 bit	RFC-1320	obsoleto
MD5	512 bit	128 bit	RFC-1321	obsoleto
RIPEMD-160	512 bit	160 bit	ISO/IEC 10118-3	buono?
SHA-1	512 bit	160 bit	FIPS 180-1	sufficiente
SHA-224	512 bit	224 bit		
SHA-256	512 bit	256 bit	FIPS 180-2	
SHA-384	512 bit	384 bit	FIPS 180-3	buoni
SHA-512	512 bit	512 bit		
SHA-2				
SHA-3	1152-576	224-512	FIPS 202 FIPS 180-4	ottimo

⚠ 1bit modificato nel msg → 50% digest diversi

⚠ Il digest però non serve contro i MITM (può modificare sia il msg originale sia il digest se il canale non è sicuro, ovvero se il digest non è protetto):



Quindi il digest deve essere protetto e lo si fa **autenticando il digest**: dimostro di essere un determinato utente, facendo **encrypting con una chiave (simmetrica tra mittente e destinatario)**, così che il destinatario farà **decrypting con la stessa chiave**, ottenendo il digest del mittente [**autenticazione basata su conoscenza di un segreto condiviso**] → solo chi conosce la K può calcolare il digest e confrontarlo. Tutto ciò è il **KEYED-DIGEST**:



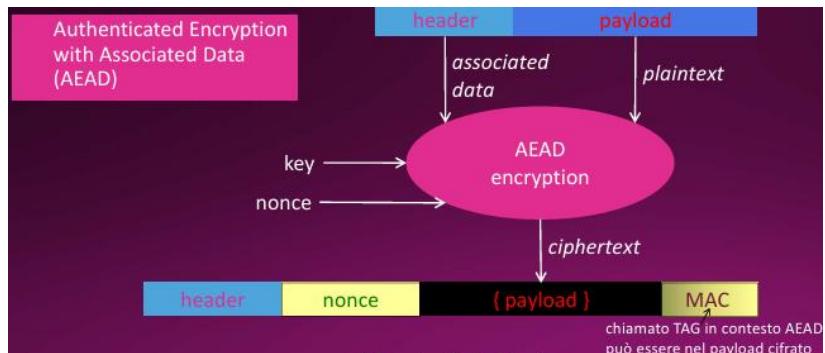
Si aggiungono dei codici ai messaggi:

- **MIC** (Message Integrity Code) → per garantire integrità dei messaggi
- **MID** (Message ID) → per evitare attacchi di tipo replay
- **MAC** (Message Authentication Code) → autenticazione

Esempio di Keyed-Digest è **HMAC** (usa anche opad e ipad che sono codici fissi ripetuti per la dimensione in byte di un blocco B). Poi abbiamo **CBC-MAC** che sfrutta un algoritmo di cifratura simmetrico a blocchi, in modalità CBC con IV nullo, prendendo come MAC la cifratura dell'ultimo blocco.

Abbiamo poi **Authenticated Encryption (AE)** [RFC 5116] che fa un'unica operazione per riservatezza + autenticazione (ed integrità); ha 1 chiave, 1 algoritmo, > velocità e meno probabilità di errori nel combinare le funzioni.

I normali schemi di cifratura sono soggetti ad attacchi **chosen-ciphertext** (modifico ciphertext; vedo se ricevente segnala errore o meno). **GCM** è il più popolare tra gli AE.



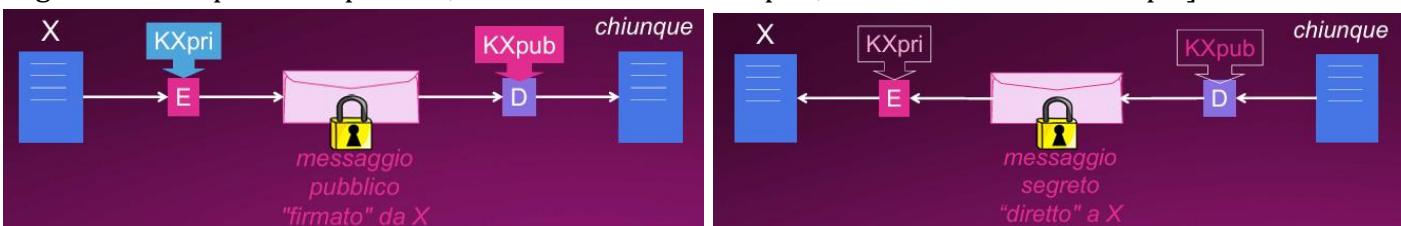
3) CRITTOGRAFIA ASIMMETRICA

Nella crittografia simmetrica, ci deve essere sempre una chiave comune simmetrica tra mittente e destinatario e questo è complicato (soprattutto la condivisione di questa); inoltre la chiave deve essere 1 per ogni coppia/gruppo di utenti, ovvero **per una comunicazione privata completa tra N persone occorrono $N * (N - 1)/2$ chiavi**, distribuite mediante distribuzione OOB (Out-Of-Band) e algoritmi per scambio di chiavi.

Perciò è stata introdotta la **CRITTOGRAFIA A CHIAVE PUBBLICA (ASIMMETRICA)**, la quale usa 2 chiavi diverse e algoritmi di encrypt/decrypt che possono anche essere diversi tra loro. Al tempo stesso sono **troppo pesanti e quindi inutili per cifrare tanti dati, quindi in realtà viene usato per distribuire le chiavi segrete (per la crittografia simmetrica) e per le firme elettroniche** [*].

Quindi le chiavi asimmetriche vengono **generate insieme a coppie**, in quanto sono complementari: le **cifrature fatte con una chiave vengono annullate usando l'altra chiave della coppia**; una delle 2 chiavi viene tenuta segreta (chiave privata), l'altra viene distribuita (chiave pubblica).

Dunque la cifratura asimmetrica fornisce **autenticazione e integrità dei dati**: il documento (in realtà un suo *digest* [riassunto]) viene cifrato con la chiave privata dell'autore; io posso usare la chiave pubblica dell'autore per "aprirla" (in questo modo è garantita l'integrità del documento ed il fatto che il documento appartenga all'autore) [dx] [a sx invece la procedura con cui posso inviare un messaggio segreto ad una persona specifica, facendo E con la sua Kpub, e lui farà D con la sua Kpri]



Le chiavi asimmetriche sono **molto più grandi di quelle simmetriche (1024-2048 bit**, quindi come dicevamo sopra, non adatte ai nostri processori per operazioni veloci) [*] e gli algoritmi a chiave pubblica sono basati su complessi algoritmi matematici da risolvere:

RSA → basato sulla **fattorizzazione in numeri primi**, ma di numeri talmente grossi (2^{1024} e 2^{2048}) che diventa computazionalmente pseudo-impossibile da risolvere; usato per segretezza e firma digitale.

Funzionamento:

- scelgo p e q numeri primi molto grandi [SEGRETI]
- calcolo $n = p \cdot q$ [PUBBLICO]
- calcolo PHI di Eulero: $\varphi(n) = (p - 1)(q - 1)$ [SEGRETO]
- scelgo l'esponente **pubblico** e tale che $1 < e < \varphi(e)$ e relativamente primo con φ
- calcolo l'esponente **privato** $d = e^{-1} \text{ mod}(\varphi)$

CHIAVE PUBBLICA = (n, e) ; CHIAVE PRIVATA = (n, d)

Plaintext: x ($< n$)

Encrypt (verifica): $c = x^e \text{ mod}(n)$

Decrypt (firma): $x = c^d \text{ mod}(n)$

⚠ Quindi gli esponenti e e d sono **intercambiabili** perché sono scelti in modo che:

$$(x^e)^d \text{ mod}(n) = (x^d)^e \text{ mod}(n)$$

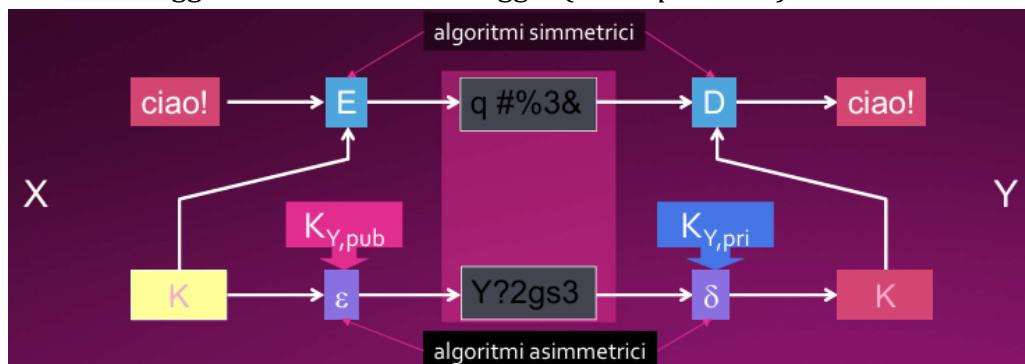
⚠ Con lo sviluppo dei **computer quantistici**, RSA diventerà vulnerabile (in quanto potranno fattorizzare numeri veloci)

⚠️ Esponenti pubblici scelti con $e = 3$ o 17 sono vulnerabili ad attacchi, mentre 65537 è il più sicuro; un attacco che veniva fatto contro i server era un attacco DDoS che inviava una chiave con molti 1 e questo generava un carico pesante contro il server

⚠️ Usare il [teorema cinese dei resti](#) velocizza x4 le operazioni di firma e decifratura con chiave privata, ma al tempo stesso se l'HW è in possesso dell'attaccante, siamo suscettibili ad attacchi di fault injection

Quindi vediamo le applicazioni della crittografia asimmetrica sopra citate:

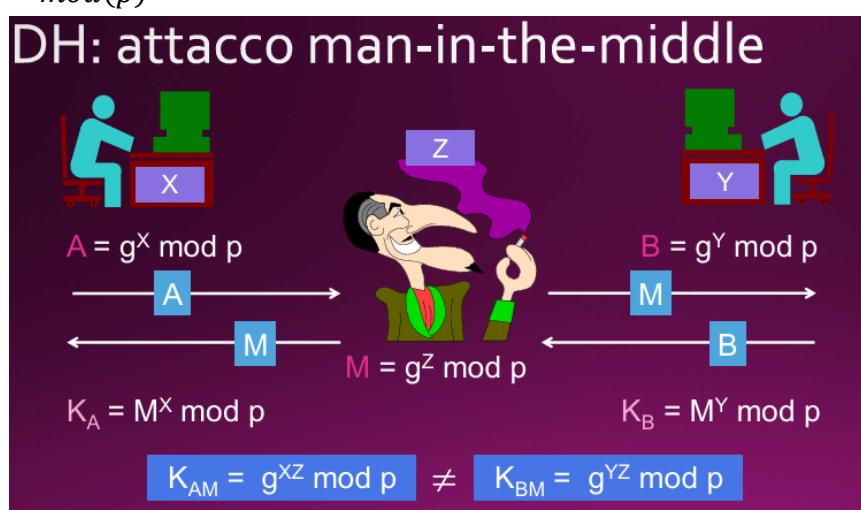
- **SCAMBIO CHIAVE SEGRETE** (simmetriche) mediante algoritmi asimmetrici → comunico una chiave di sessione o di messaggio K insieme al messaggio (*enveloped data*)



- **DH (Diffie-Hellman)** → 1° algoritmo a chiave pubblica inventato, usato per key agreement; resistente allo sniffing, ma se l'attaccante può manipolare i dati allora è possibile un attacco MITM.

Funzionamento:

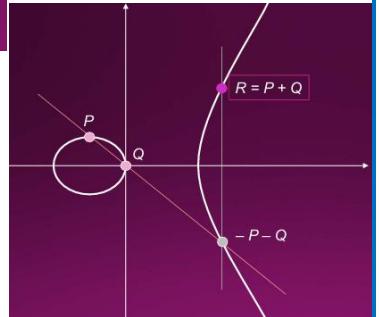
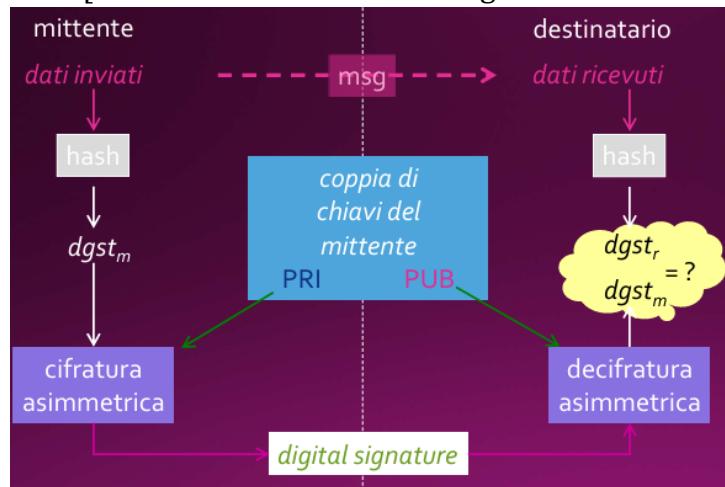
- A e B scelgono 2 interi pubblici: g (generatore), p (n° primo grande)
- A, scelto a caso un intero grande $x > 0$, calcola $X = g^x \text{ mod}(p)$
- B, scelto a caso un intero grande $y > 0$, calcola $Y = g^y \text{ mod}(p)$
- A e B si scambiano X e Y
- A calcola $K = Y^x \text{ mod}(p)$
- B calcola $K' = X^y \text{ mod}(p)$
- ma $K = K' = g^{xy} \text{ mod}(p)$



- **Autenticazione** → si invia anche un **digest** (cifrato con la chiave privata del mittente):

- firmatario S: $H = enc(S_{Kpri}, \text{hash}(M))$
- trasmissione: $M || H$
- verificatore V: $X = dec(S_{Kpub}, H)$
- verifica: $\text{if}(X == \text{hash}(M)) \text{ then OK else ALARM!}$

Chi conosce la K_{pub} può confrontare il digest trasmesso con quello calcolato sui dati ricevuti → questa è la **FIRMA DIGITALE** (quindi permette di dimostrare l'integrità, senza dover rivelare a tutti la chiave simmetrica [cosa che andrebbe fatta in algoritmi simmetrici])



CRITTOGRAFIA A CURVE ELLITTICHE (ECC) → introduce operazioni matematiche complesse (quindi la crittografia è la stessa, ma basata su una aritmetica più complessa, basata sulle curve ellittiche), equivalenti all'inverso del logaritmo discreto su una curva ellittica. Usata in firma digitale (**ECDSA**), key agreement (**ECDH**, **ECMQV**) e key distribution (**ECIES**)

ECDH → funzionamento:

- A e B scelgono la stessa curva ellittica j ed un suo punto G
- A, scelto a caso un intero grande x , calcola $X = xG$
- B, scelto a caso un intero grande y , calcola $Y = yG$
- A e B si scambiano X e Y
- A calcola $K = xY$
- B calcola $K' = yX$
- ma $K = K' = xyG$

⚠ ECC dipende dalle curve usate (non tutte sono sicure); le sceglie il NIST

KDF (Key Derivation Function) → visto che gli utenti preferiscono le password e non le chiavi crittografiche random, esiste il KDF tale che $K = KDF(P, S, \text{iterazioni})$ con P = password, S = salt, $\text{iterazioni} = n^{\circ}$ di iterazioni della funzione base; sono fatte per **ottenere una key crittografica da una password, ma in modo lento apposito** (tramite **hash crittografico**) in modo che l'attaccante ci debba mettere un tempo altissimo per scoprirla (esempi sono PBKDF₂ e HKDF)

⚠ Sono attaccabili da GPU e ASIC; per questo è stato introdotto un nuovo requisito: è richiesta tantissima RAM per derivare una password, in modo da bloccare GPU e ASIC (quindi non richiedono tanta potenza computazionale, ma tanto spazio di memoria)

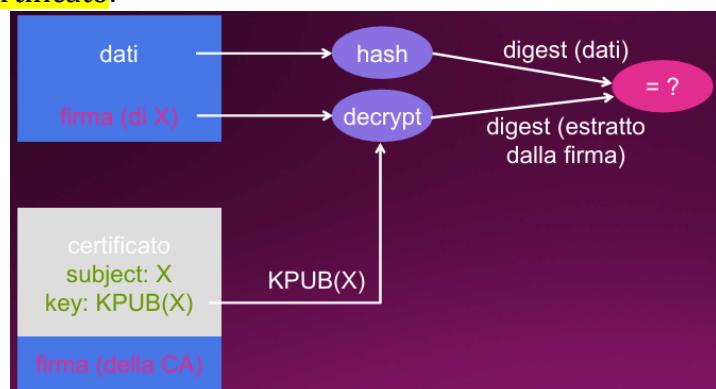
POST-QUANTUM CRYPTOGRAPHY (PQC) → DH è risolvibile su computer quantistici, ma l'algoritmo di **Shor** permette di fattorizzare numeri interi, velocissimo su un computer quantistico (quindi RSA, DSA e ECC diventerebbero deboli). Serve quindi una **nuova classe di problemi che non siano risolubili in tempo polinomiale nemmeno da un computer quantistico** (es. symmetric key quantum resistance, hash-based cryptography) [es. CRYSTALS-Kyber]

⚠ Crittografia simmetrica e funzioni di hash non sembrano soffrire quantum (es. algoritmo di Grover)

CERTIFICATI DIGITALI → la distribuzione delle chiavi per crittografia asimmetrica funziona se la chiave privata non è mai divulgata e la chiave pubblica è distribuita il più ampiamente possibile; **ma chi garantisce la corrispondenza tra chiave pubblica e identità della persona?** Soluzione: **distribuzione della chiave pubblica in un certificato digitale a chiave pubblica (certificato d'identità digitale)**

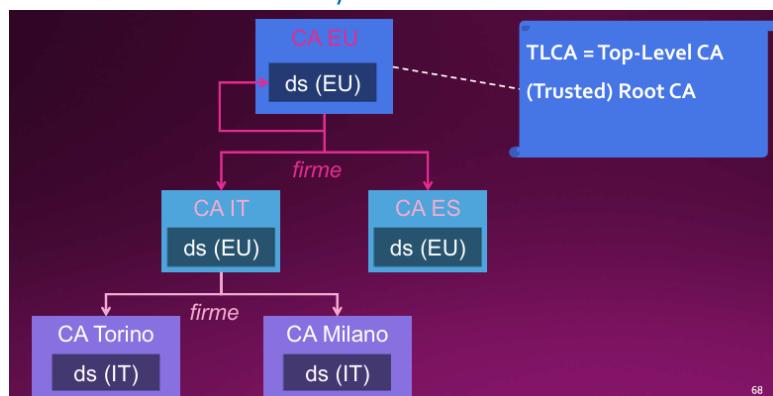
CERTIFICATO A CHIAVE PUBBLICA = struttura dati per legare in modo sicuro una chiave pubblica ad alcuni attributi (lega una chiave pubblica ad un'identità); viene firmato in modo elettronico dall'emittente (**autorità di certificazione – CA [Certification Authority]**), con **scadenza temporale**, ed è revocabile sia dall'utente sia dall'emittente. La CA deve tenere traccia dei vari certificati (anche persi e revocati): dunque parliamo di **PKI (Public-Key Infrastructure)**, ovvero l'infrastruttura tecnica e organizzativa che serve alla creazione, distribuzione e revoca dei certificati a chiave pubblica. I formati per certificati a chiave pubblica sono **X.509 v3** (inventato direttamente con lo stack ISO-OSI), non X.509 (**PGP, SPKI**) e **PKCS#6** (RSA, in parte compatibile con X.509, ma obsoleto).

Verifica di una firma/certificato:



Quindi se devo verificare l'autenticità di un certificato a chiave pubblica firmato da CA_1 , ho bisogno del certificato della chiave di CA_1 firmato da CA_2 ; però la firma di CA_2 deve anch'essa essere verificata e dunque occorre il certificato della chiave di CA_2 firmato da CA_3 etc...

Dunque occorre una **gerarchia di certificazione/distribuzione**:



Ma come mai io non ho mai esplicitamente detto "Mi fido di questo certificato"? Perché questo **lo fanno direttamente i browser**.

Come si **revoca un certificato** prima della sua scadenza naturale? Può avvenire su richiesta del titolare e autonomamente dall'emittitore. Quando si valida una firma digitale bisogna verificare che il certificato fosse **valido all'atto della firma** (verifica a carico del ricevente). Meccanismi di revoca:

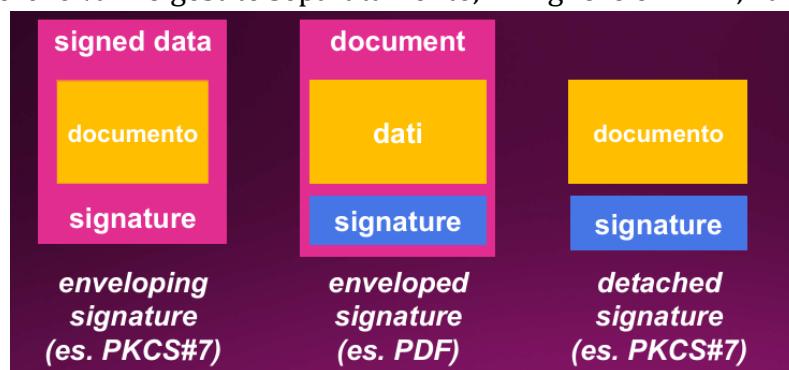
- **CRL** (Certificate Revocation List) → elenco di numeri di serie di certificati revocati (firmato dalla CA) e motivazione/statuto della revoca
- **OCSP** (On-line Certificate Status Protocol) → risposta su 1 certificato (firmata dal server) indicante lo stato di revoca di un certificato solo all'istante attuale (dice se revocato o no, ma non quando)

PRESTAZIONI e ACCELERATORI → le prestazioni crittografiche non sono un problema su client, ma sui server, sui nodi di rete e su dispositivi embedded: per ciò si usano **ACCELERATORI CRITTOGRAFICI**

⚠ **PSE (Personal Security Environment)** = ogni utente dovrebbe proteggere la sua chiave segreta e i certificati delle CA che ritiene trusted/autentici; PSE **software** = file cifrato contenente la chiave privata; PSE **hardware** = può essere passivo (memorizza le chiavi) o attivo (fa azioni crittografiche)

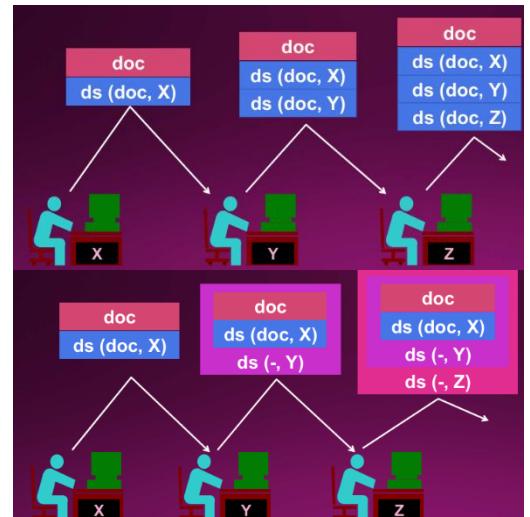
Alcuni **PSE hardware** sono **Smart-card crittografica** (smart card con chip con piccola memoria interna e implementazione delle primitive crittografiche) [< \$] e **HSM (HW Security Module**; acceleratori crittografici per i server con memorizzazione sicura della chiave privata e azioni crittografiche)

DOCUMENTI ELETTRONICI SICURI (documento elettronico = rappresentazione elettronica di un doc; documento digitale = rappresentazione digitale) [integrità, autenticazione dell'autore, usabilità e leggibilità, archiviazione a lungo termine] → diversi formati di documenti firmati (il peggiore è quello delle firme staccate perché vanno gestite separatamente; il migliore è il PDF; l'altro ha bisogno di sw):



Le **firme multiple** possono avvenire in 2 tipi:

- **PARALLELE/INDIPENDENTI** → veloce e parallelo, ma non estrema sicurezza (perché ognuno valida solo il documento passato)
- **SEQUENZIALI/GERARCHICHE** → incapsulato e lento, ma estrema sicurezza (perché ognuno valida tutto il pacchetto incapsulato [doc + firma precedente])



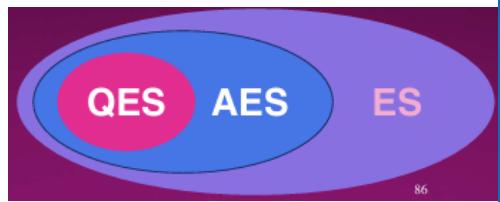
Time-stamping = prova che alcuni dati siano stati creati prima di una certa data; validati dalla **TSA** (Time-Stamping Authority):



QC (Qualified Certificate) → certificato a chiave pubblica rilasciato per certificare l'identità di una persona (indica che è un QC, indica nome/pseudonimo dell'identità, include attributi [uso]) [**RFC-3739**]

ES (Electronic Signature):

- **AES (Advanced ES)** → identificazione univoca del firmatario, memorizzazione delle chiavi e integrità/non-forgiabilità dei dati
- **QES (Qualified ES)** → firma digitale basata su un QC, creata con un dispositivo per la generazione di firme sicuro (soddisfa requisiti legali, come le firme manuali) [usabile come prova in tribunale]



⚠ **Formati di ES:** CMS (CAdES), XML (XAdES), PDF (PAdES)...

⚠ Firmare un documento con delle macro (quindi delle variabili cambiabili) non è una buona idea!

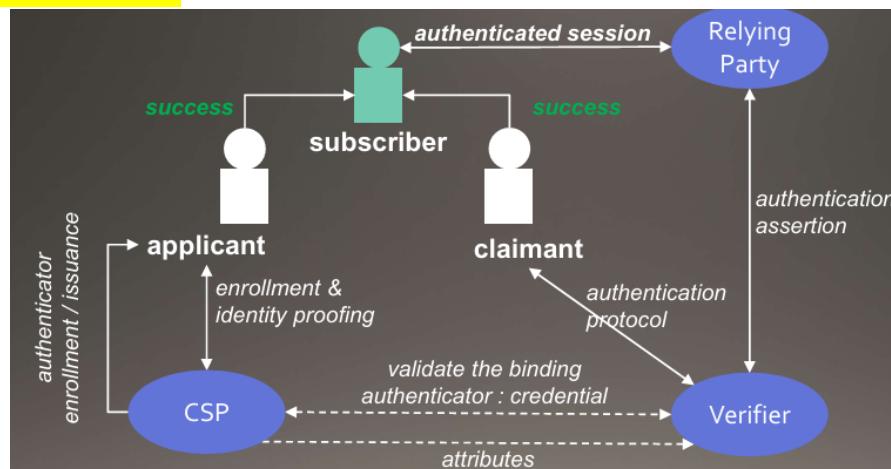
4) AUTENTICAZIONE (authN/authC)

Autenticazione = dimostrare/dichiarare l'identità di un attore (essere umano, software, hardware...).

Come dimostro un'affermazione nel mondo informatico?

- **Knowledge (conoscenza)** = qualcosa che solo l'utente conosce (es. PIN, password...)
- **Ownership (possesso)** = qualcosa che l'utente possiede [*authenticator*] (es. token, smart card...)
- **Inherence (essenza)** = qualcosa che l'utente è (es. impronta digitale...)

Digital Authentication Model:



Le entità logiche (anche combinate tra loro) coinvolte nell'authN sono:

- **RP (Relying Party)** = richiede e riceve un'asserzione di autenticazione dal Verifier per certificare l'identità e gli attributi utente
- **CSP (Credential Service Provider)** = registra le credenziali utente; verifica gli attributi associati
- **Verifier** = esegue un protocollo di autenticazione per verificare l'asserzione di autenticazione

Un'autenticazione forte (**strong authN**) è basata su **2 o + fattori** tra knowledge, ownership e inherence, dove gli elementi usati devono essere mutualmente indipendenti ed almeno 1 degli elementi deve essere non-riusabile (es. OTP) e non-replicabile.

Autenticazione degli utenti →



Vediamo possibili metodi di autenticazione degli utenti e le loro problematiche:

- **PASSWORD** (ripetibili, cioè normali pwd): il segreto è la password dell'utente; il client crea e trasmette la prova (ovvero la password); il server memorizza e valida la prova (hashed digest)



Problemi = generazione e memorizzazione della pwd, trasmissione al server (confidenzialità); le password hashed possono essere rubate dal server (*data breach*) → l'attaccante non può invertire la funzione di hash (e tornare al plaintext), ma può fare *dictionary attack* (file dizionario con tantissime password rubate vengono usate in attacchi **brute force** [così non si provano tutte le combinazioni, ma quelle più usate/diffuse]) [usando GPU, la pre-computazione riduce tanto i tempi e si possono poi provare tantissime password in parallelo, anche usando tool di supporto]

Altri problemi sono:

- **password duplication** (alcuni utenti usano la stessa pwd ovunque, quindi data breach = perdere la password ovunque!)
- **phishing** e spoofing
- **MITM**

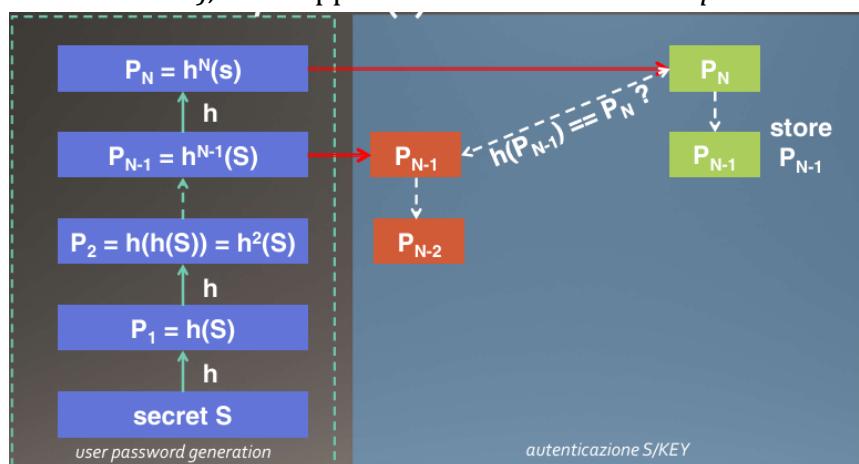
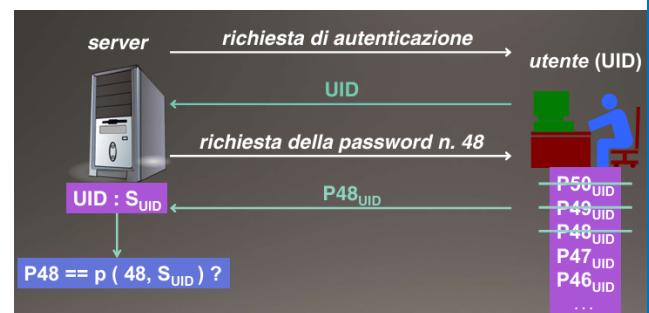
⚠ Il server in realtà non memorizza l'hashed password, ma il **salt randomico** dell'utente; in questo modo, il salt differenzia i risultati dell'hash dello stesso messaggio, ma per utenti diversi. In questo modo si contrastano le "*rainbow table*" e quindi gli *attacchi dizionario* (tabelle con moltissime hashed password precompilate, ma grazie al salt, non rimangono costanti, quindi poco utili). Dunque vengono memorizzate le triple $\{UID, \text{salt}_{UID}, HP_{UID}\}$

Best Practice = password lunghe, non in dizionario, cambiate frequentemente, password wallet (utili a memorizzare una pwd diversa per ogni sito, ma single-point-of-failure)

- **OTP** (non-ripetibili, **One-Time Password**): pwd valida per 1 solo uso; immune allo **sniffing**, ma soggetto a **MITM** (serve autenticazione del Verifier) ed è **difficile fornire** le pwd ai Subscriber, oltre che è **difficile inserire** le password (costituite da caratteri random infatti).

Per **fornire OTP agli utenti**, su postazioni intelligenti (es. app delle banche o authenticator), vengono calcolate dinamicamente.

Il 1° sistema usato è **S/KEY** (l'hash della pwd 99 è la pwd 100; quindi si butta via la 100 e si tiene la pwd 99 per fare il confronto...), che è appunto un sistema "*Pre-Computed OTP*":



Quindi nella pratica: utente genera un segreto S_{ID} e poi calcola N OTP (con formula $P_1 = h(S_{ID})$, ... $P_N = h(P_{N-1})$); il verifier invece memorizza l'ultima password P_N e le usa indirettamente a ritroso per l'autenticazione.

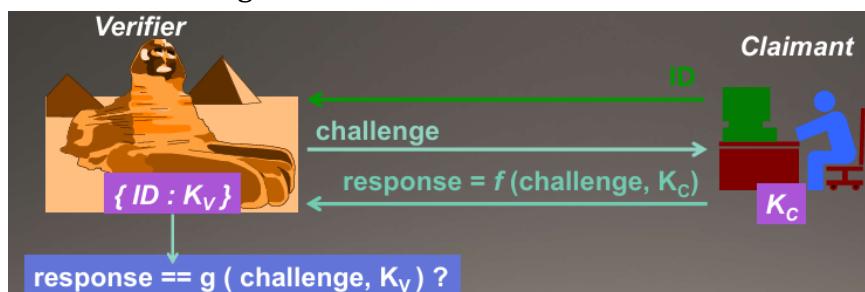
Poi si è passati a **Time-based OTP** [pwd dipende dal tempo e dal segreto dell'utente $\rightarrow p(ID, t) = h(t, S_{ID})$]; qui il problema è il costo del dispositivo ed il fatto che l'orologio interno al sistema è poco preciso (quelli precisi costano tanto). Inoltre anche se si spende di più per orologi precisissimi, dopo qualche hanno si desincronizza (quindi va cambiato/ricomprato dall'ente fornitore) [oggi infatti si usano *OTP sullo smartphone*]:



Il problema di questo servizio era anche il fatto che i segreti di tutti gli utenti dovevano essere sul **server** (quindi un attacco ad esso era catastrofico) in quanto è il server a fare il calcolo di S_{ID} e t . Un problema sono anche gli **attacchi temporali** (si usava l'orologio del server e del client, modificando l'ora per arrivare/tornare al momento dell'OTP e si usavano gli OTP), ma il problema principale per cui questo sistema è stato abbandonato è il **costo**.

⚠ Altro problema è che non si possono fare più operazioni insieme (perché OTP legato al tempo), quindi è stato poi anche inventato un sistema "**Event-based OTP**" con vantaggio che l'OTP viene cambiato con l'avanzare del **contatore** dell'utente ("claimant" o richiedente) e non con il tempo (quindi non ha problemi di sincronizzazione, ma eventuali pressioni ripetute involontarie non hanno favorito l'usabilità).

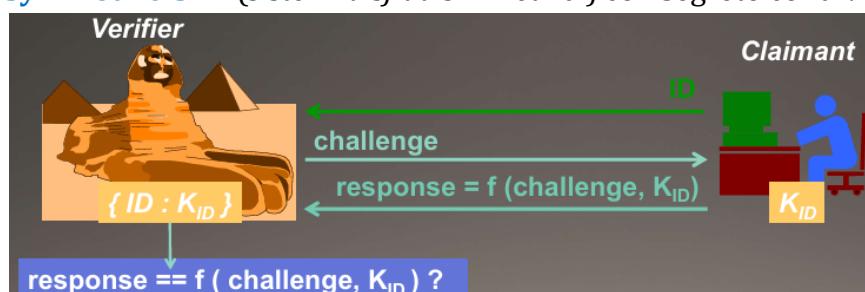
- **CRA (Challenge-Response Authentication)**: una **sfida** viene inviata al claimant, il quale risponde con una soluzione calcolata usando segreto e sfida; poi il verifier confronta la risposta con la soluzione calcolata tramite un segreto correlato al claimant:



La sfida deve essere **non-ripetibile** (per evitare attacchi replay) e **non-invertibile** (hash):

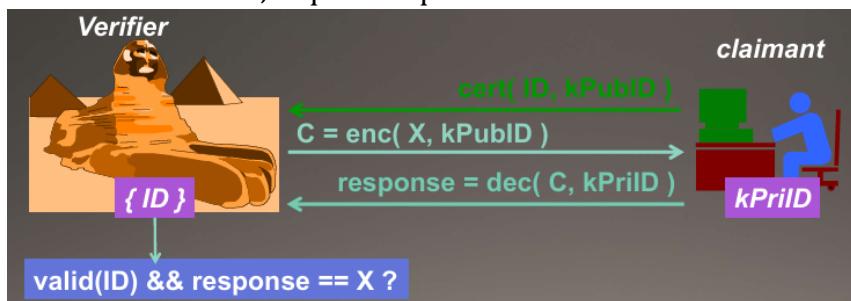
$$K_C = f^1(response, challenge)$$

Ci sono sistemi **Symmetric CRA** (*sistemi a sfida simmetrici*) con segreto condiviso + hash:



⚠ Il problema di questi sistemi è che la K_{ID} deve essere nota in chiaro sull'authN server, ma viene risolto con alcuni meccanismi come **SCRAM** (usa l'hash di password sul verifier)

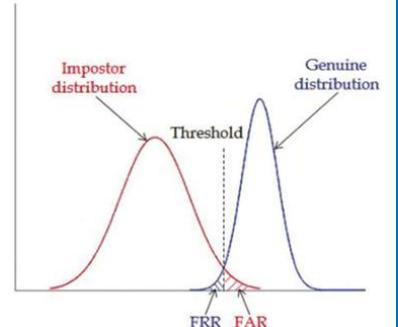
Ci sono poi anche sistemi **Asymmetric CRA** (la sfida viene generata cifrando un random nonce X con la chiave pubblica del claimant, il quale risponde inviando X in chiaro decifrando con la chiave privata):



⚠ È il meccanismo più forte, non richiede memorizzazione di segreti sul verifier, implementato in quasi tutti i protocolli di sicurezza; è quello su cui si basano le *passkey*; è però lento

⚠ Idealmente, se ci mandassero un contratto fasullo su cui è stato fatto da hash dall'attacker, noi risponderemmo applicando la chiave privata su questo, facendo una firma digitale su un doc fasullo

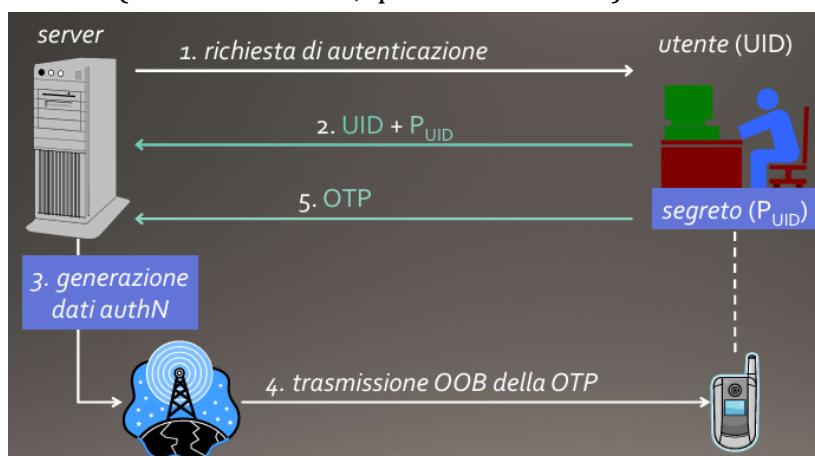
- **Sistemi Biometrici** (Autenticazione di Esseri Umani): usato per verificare di interagire con esseri umani (**non con i bot** [processi automatici]; es. CAPTCHA); un metodo per fare ciò è la **BIOMETRIA** (es. volto, retina, impronta digitale...) i cui sistemi sono caratterizzati da **FAR** (False Acceptance Rate) e **FRR** (False Rejection Rate) → più il dispositivo è di fascia alta, più è improbabile che un estraneo passi la biometria al posto nostro (*FAR*) e più è improbabile che io non venga riconosciuto (*FRR*).



⚠ Problemi: *accettazione psicologica* (paura di essere schedati o di danneggiamento alla retina), *privacy*, impossibile sostituire il “volto” se viene scoperto e problemi di API/interoperabilità

- **Multi-Factor Authentication (MFA)**:

- **Out-of-Band OTP** (solitamente SMS, quindi attaccabile):

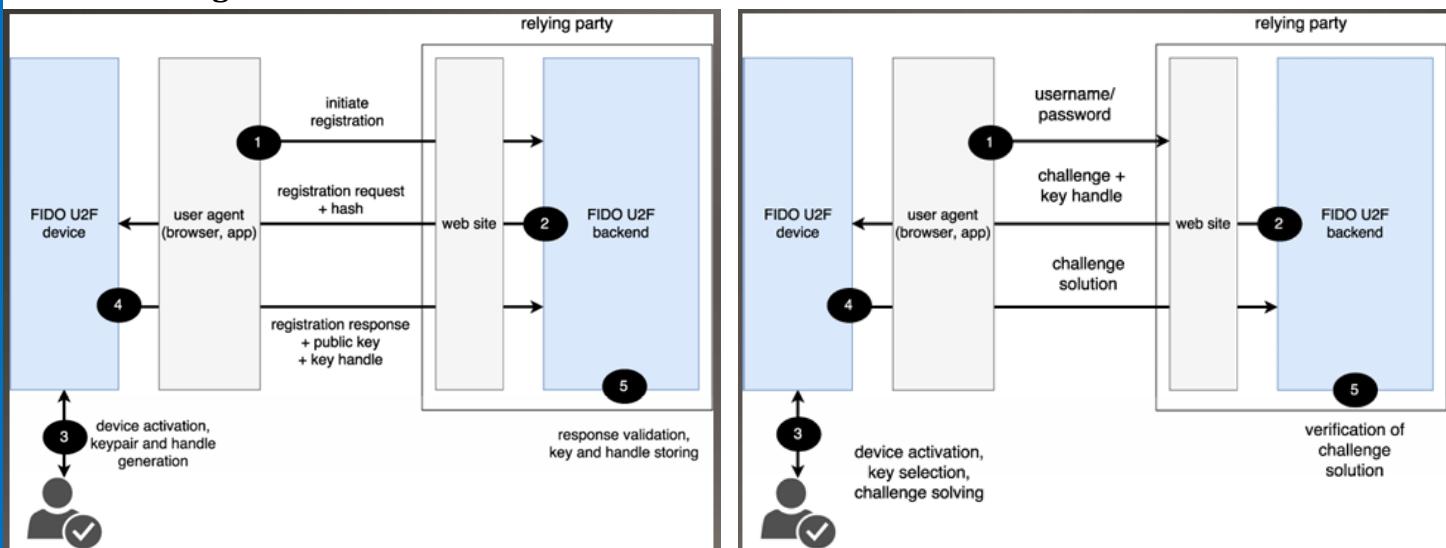


- **Dispositivo personale dell'utente (biometrico)** → durante la registrazione viene scambiato un segreto (che viene salvato nel dispositivo + il verifier registra dati dello smartphone [marca + modello] ed il segreto); oltre al 1° fattore di autenticazione viene chiesto di generare un OTP da un'app specifica [es. *authenticator*] in modo da garantire prova di possesso del dispositivo + prova di conoscenza del segreto scambiato

- **Passwordless authN e Passkey:** le **PASSKEY** sono un tentativo di sostituire le password con **CRA asimmetrici**, ma senza lasciare agli utenti la sbatta di gestire le credenziali e evitando phishing e tracciamento degli utenti. Sono un sistema di authN *cross-platform* che vengono spesso usato come **2° fattore** in un MFA. Ci sono 2 fasi:

- **Registrazione per un singolo sito** → invece di salvare una pwd sul sito, viene creata una **coppia di chiavi asimmetriche, associate localmente in un DB interno alla URL del sito su cui ci si sta registrando** (il sito salva la mia KPUB generata quando mi registro sul sito)
- **Login su sito** su cui mi sono già registrato → avendo già una coppia di chiavi pronte, **dopo aver riconosciuto la URL del sito, si dimostra di possedere la componente privata** mediante CRA asimmetrico

FIDO (*Fast Identity Online*) ha creato lo standard industriale per questa passwordless authN; usa dispositivi personali capaci di fare crittografia asimmetrica (per **rispondere alle sfide asimmetriche** al momento del login sui siti, e per **firmare digitalmente**). Vediamo a sx la **FIDO registration** e a dx il **FIDO login**:

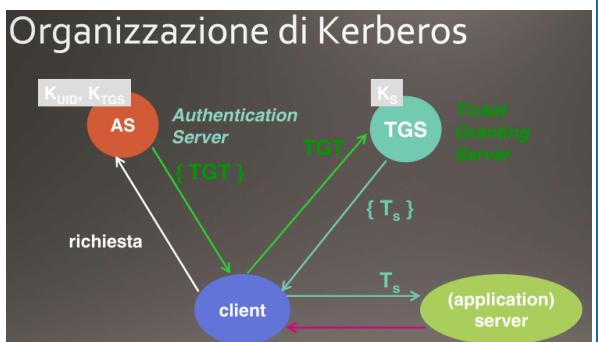


⚠ Altre caratteristiche di FIDO → **biometria** (metodo di authN locale per abilitare le chiavi FIDO salvate sul device utente), **transazioni sicure** (la risposta alla sfida è una firma digitale di un testo relativo alla transazione), **FIDO server/backend** (per abilitare FIDO su un server applicativo) e **FIDO client** (per creare e gestire credenziali FIDO sul device utente)

Quindi si garantisce **authN forte** (con **crittografia asimmetrica**), **non** ci sono **terze parti** incluse nel protocollo, **nessun segreto conservato nel server, no phishing...**

Zero-Knowledge Proof (ZKP) → insieme di procedure dove il verifier non conosce il segreto del claimant (quindi **non è richiesto che l'utente condivida il suo segreto con il server**); quindi in pratica il verifier non sa il segreto effettivo, ma ha solo alcune info su di esso: fa molte domande booleane all'utente e se risponde correttamente può procedere [ZKPP = ZKP su password; ZKA = ZKP authN]

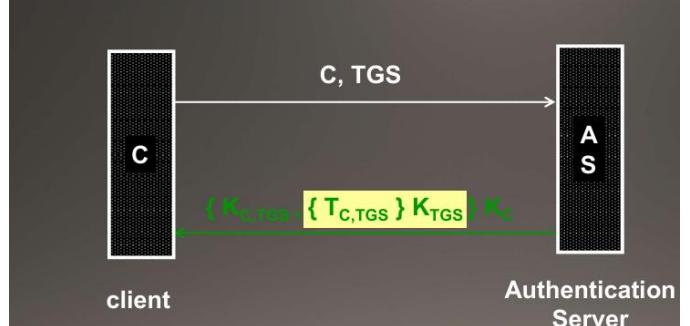
SSO (Single-Sign On) = **per esempio quello del Poli**: l'utente ha un'unica credenziale per autenticarsi 1 volta e poi accede a tutti i servizi del sistema, perché **ogni servizio viene avvisato da quello che ha già verificato l'authN che l'utente è autenticato correttamente**. Parliamo di **SSO fittizio** (es. password wallet) e **vero SSO** (authN multi-app [Asymmetric CRA e SSO multi-dominio con SAML]). Un esempio è **Kerberos**: sistema di authN basato su un *trusted third-party*.



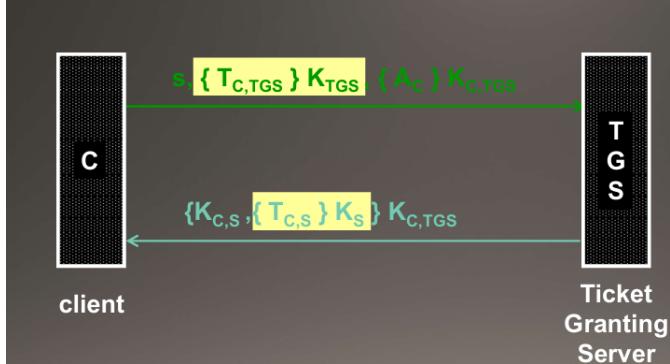
Come funziona Kerberos?



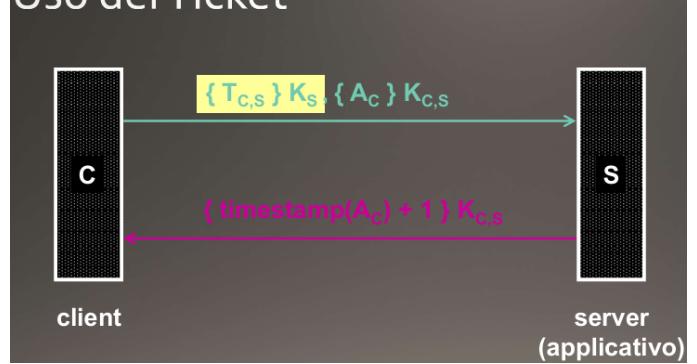
Richiesta del TGT



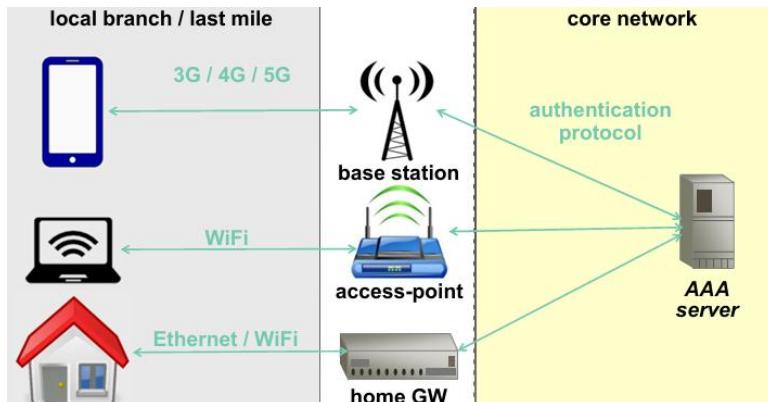
Richiesta del Ticket



Uso del Ticket



5) ACCESSO ALLA RETE + MINACCE e MITIGAZIONI per reti IP

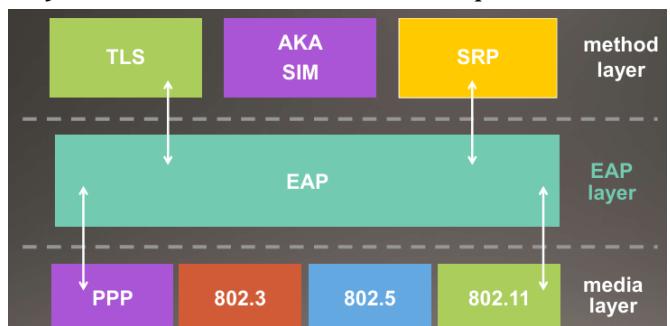


PPP è un protocollo per **incapsulare pacchetti di rete L_3** (IP) e **trasportarli su collegamenti punto-punto** (reali [RTC], virtuali [L_2 con PPPoE oppure L_3 con L_2 TP su UDP/IP]). **3 fasi in sequenza:**

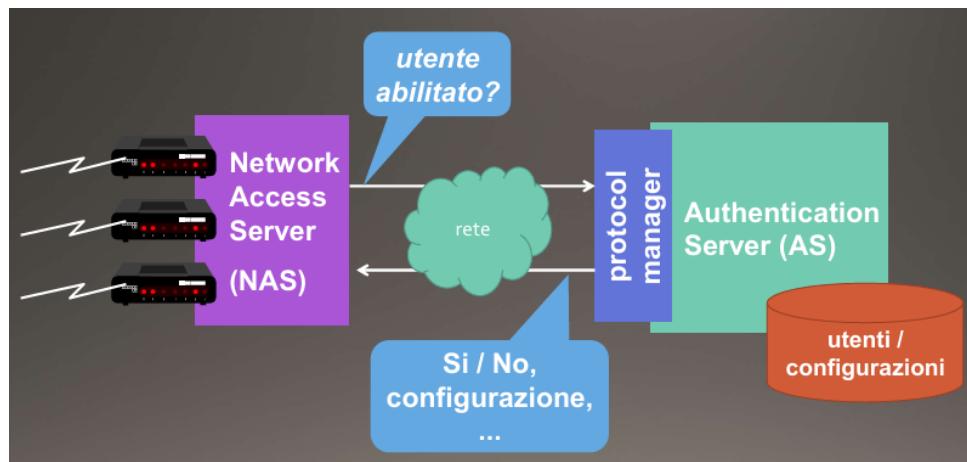
- **LCP** (Link Control Protocol) = negoziazione dei parametri di connessione
- **Autenticazione (opzionale)** = riconoscimento degli endpoint [a noi interessa questo (*)]
- **L_3 encapsulation (IPCP)** = trasferimento dati

L'autenticazione degli accessi in rete (*) avviene con diversi **authentication protocol**:

- **PAP** (Password Authentication Protocol) = pwd utente inviata in chiaro [obsoleto]
- **CHAP** (Challenge Handshake Authentication Protocol) = sfida asimmetrica basata su pwd utente
- **EAP (Extensible Authentication Protocol)** = quello **usato ora**; si aggancia a meccanismi esterni (sfide, OTP, TLS) ed infatti è un **framework flessibile a livello data-link (L_2) a cui si possono aggiungere nuove tecniche** [le prime aggiunte furono standardizzate] [quindi come una sorta **middleware**]



Autenticazione per accesso in rete:

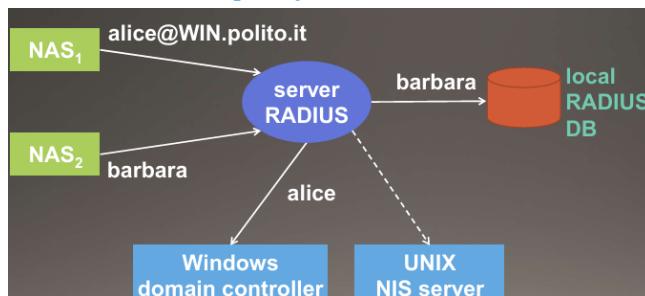


I produttori di **NAS** (Network Access Server) dicono che la sicurezza richiede 3 funzioni:

- Autenticazione → il claimant viene autenticato in base alle sue credenziali
- Autorizzazione → si decide se l'azione o il tipo di accesso è permesso al claimant
- Accounting → si tracciano le risorse consumate (per farle poi pagare)

L'Authentication Server (AS) ricopre queste 3 funzioni parlando con 1 o + NAS tramite **protocolli di autenticazione via rete** come:

- **RADIUS** = più diffuso; ha funzionalità di proxy verso altri AS:



Ha amministrazione e accounting *centralizzato*. Applica lo schema **client-server tra NAS e AS**. I suoi pacchetti hanno formato **code + identif + length + autenticatore** (autentica la risposta del server [evitando replay] e maschera la pwd) + attributi. I suoi pacchetti sono:

- ACCESS-REQUEST (con credenziali di accesso)
- ACCESS-REJECT (accesso negato)
- ACCESS-CHALLENGE (sfida)
- ACCESS-ACCEPT (accesso consentito)

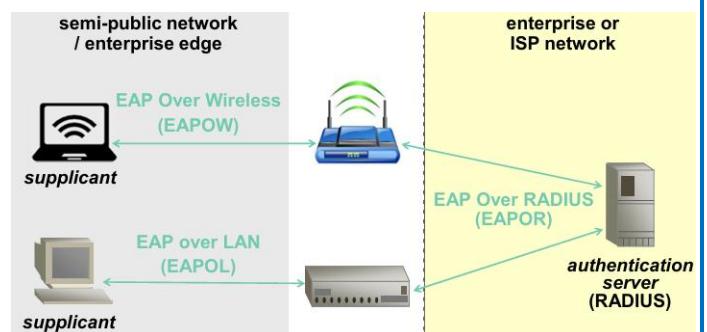
Funziona con PAP, CHAP e EAP; ha attributi TLV soliti (quindi Type-Length-Value).

RADIUS ha protezione da sniffing, spoofing e creazione di risposte false, replay di risposte, availability e resistenza a DoS. Ma al tempo stesso usa MD5, quindi **debole**.

- **DIAMETER** = evoluzione di RADIUS, ma con enfasi su **roaming tra ISP diversi** e elevata attenzione alla **sicurezza** (impone l'uso di canali sicuri tra client e server mediante IPsec o TLS)

- **TACACS+** => RADIUS, ma proprietario (< diffuso)

⚠ IEEE 802.1X ha *port-based network access control* (authN L_2), framework per autenticazione e key-management. Il vantaggio è che sfrutta L_7 per l'implementazione dei meccanismi di sicurezza (quindi perfetta integrazione con AAA, NIC e NAS).



MINACCE e MITIGAZIONI per reti IP

A quale **livello** di rete è meglio realizzare la sicurezza? **Più si sale** nello stack OSI, più le funzioni saranno **specifiche ed indipendenti** dai livelli sotto, ma più saranno possibili **attacchi DoS**; **più si sta in basso**, più sarà facile **espellere gli intrusi**, ma i **dati** su cui basare se si ha davanti un “intruso” saranno più **scarsi**.

Protezioni a livello fisico (**L1**) sono comuni solo a livello militare. Esistono protezioni a livello data/link (**L2**), ma funzionano solo in segmenti di rete con stessa tecnologia L2 (quindi limitato). Quindi si arriva a livello rete (**L3, IP**), dove si può avere **protezione e2e per reti omogenee** a livello logico (es. IP) ed è possibile creare una VPN per proteggere anche solo parte del path.

VPN (Virtual Private Network) = tecnica hw/sw per creare una rete privata, usando canali condivisi.

VPN mediante tunnel IP sicuro: i router incapsulano i pacchetti di rete in altri pacchetti (IP in IP, IP over MPLS ...), ma prima di essere incapsulati i pacchetti di rete vengono protetti con:

- **MAC** (per *integrità* e *authN*)
- **cifratura** (per *riservatezza*)
- **numerazione** (per *evitare replay*)

⚠ Infatti, se gli algoritmi crittografici sono forti, l'unico attacco possibile è bloccare le comunicazioni (**Secure VPN**)

IPsec → architettura IETF per sicurezza L3 (sia in IPv4, sia in IPv6) che permette di creare **VPN** su reti **non fidate** e fare **sicurezza e2e**. Definisce 2 formati:

- **AH (Authentication Header)** = integrità, authN (identificazione del mittente), no replay
- **ESP (Encapsulating Security Payload)** = riservatezza (cifratura dei dati) + **AH**

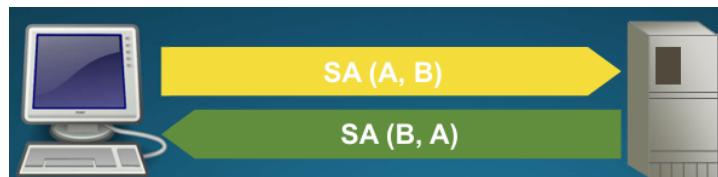
Come funziona la spedizione di IPsec? [IKE*]



Dall'immagine si vedono i **database locali** IPsec:

- **SPD (Security Policy Database)** = contiene **security policies** da applicare ai vari tipi di comunicazione
- **SAD (Security Association Database)** = elenco delle SA attive e delle loro caratteristiche (algoritmi, chiavi, parametri)

Cos'è una **Security Association (SA)**? È una **connessione logica unidirezionale protetta tra 2 sistemi IPsec**; ad ogni SA sono associabili caratteristiche di sicurezza diverse. Servono 2 SA per avere protezione completa di 1 canale bidirezionale

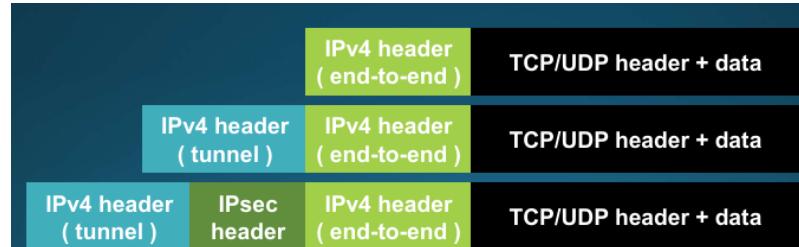


IPsec può essere in:

- **transport mode** → usato per fare **sicurezza e2e** (quindi usato dagli **host**, non dai gateway): leggero, ma non protegge i campi variabili:



- **tunnel mode** → usato per **fare VPN dai gateway**; pesante, ma protegge anche i campi variabili:

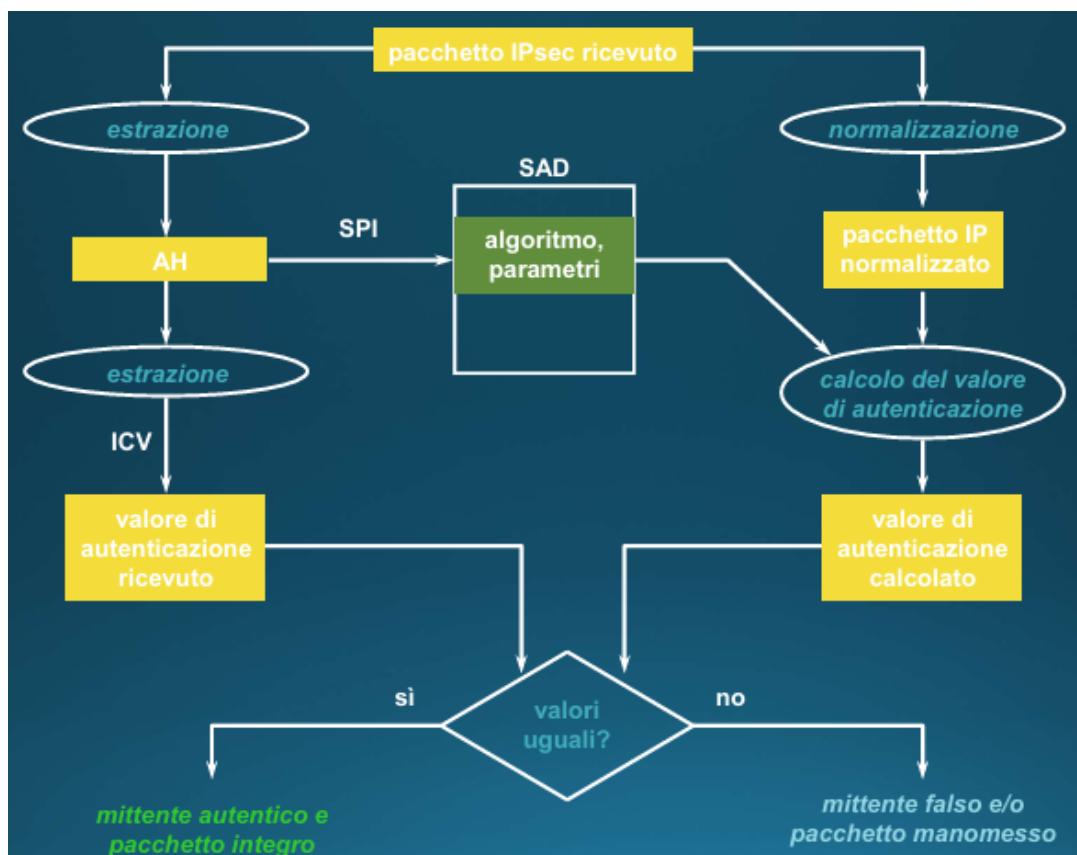


Vediamo quindi i 2 sotto-protocolli di IPsec (prima citati):

1. **AH (Authentication Header)** → **integrità dati, authN mittente e protezione da replay attack**; keyed-MD5 (con opzionale keyed-SHA-1 all'inizio), ora invece HMAC-MD5, HMAC-SHA-1

⚠ **Normalizzazione** per AH:

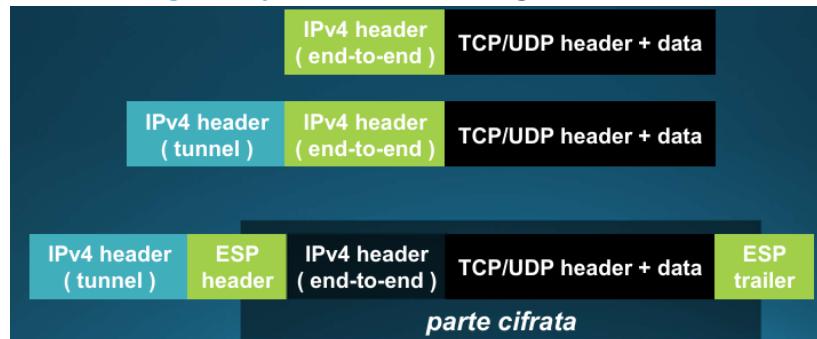
- azzerare il campo TTL/Hop Limit
- se il pacchetto contiene un *Routing Header*, allora:
 - fissare il campo destinazione all'indirizzo del destinatario finale
 - fissare il contenuto del routing header al valore che avrà a destinazione
 - fissare il campo Address Index al valore che avrà a destinazione
- azzerare tutte le opzioni che hanno il bit C (*change en route*) attivo



2. **ESP (Encapsulating Security Payload)** → all'inizio solo **riservatezza**, poi aggiunta anche **authN** (ma esclude l'header IP), quindi non dà la stessa copertura di AH [ma tanto non serve in caso di tunnel]; per questo riduce la dimensione del pacchetto e risparmia 1 SA). ESP in:
- transport mode** = usato dagli **host**, non dai gateway; non nasconde l'header



- tunnel mode** = usato dai **gateway**, nasconde anche gli **header**



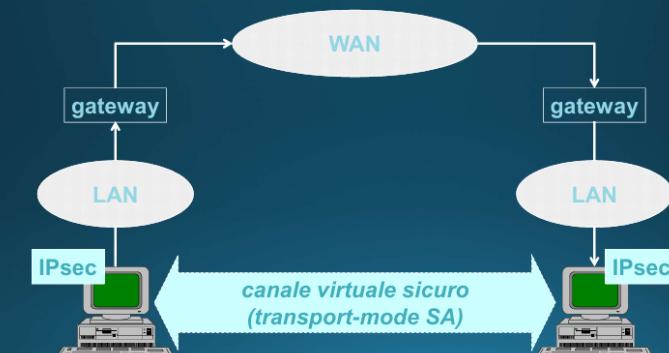
⚠ Pacchetti ESP-DES-CBC hanno tutto autenticato, mentre solo payload e padding criptati

⚠ IPsec ha **protezione da replay**: quando si crea una SA, il mittente inizializza **sequence_number = 0**; quando si invia un pacchetto, **sequence_number++**; quando si raggiunge il **sequence_number == 2³² - 1**, si negoziava una nuova SA (contro il principio di indipendenza dei datagram a L3, ma al tempo stesso garantisce questo meccanismo)

⚠ **AH è opzionale, ESP obbligatorio.** C'è supporto multicast da singola sorgente e per authenticated encryption (**AEAD**)

Vediamo quindi in **riassunto di tutte le possibili configurazioni** (c'è anche **e2e + basic VPN insieme**):

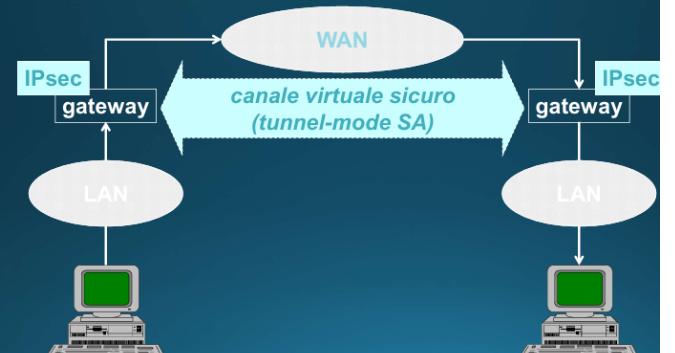
End-to-end security



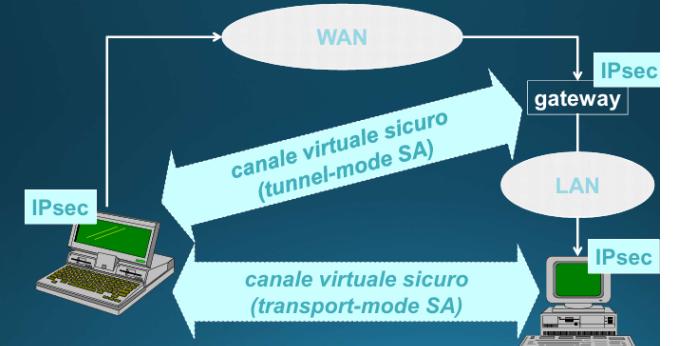
Secure gateway



Basic VPN



Secure remote access



IKE (Internet Key Exchange) [★] → componente fondamentale di IPsec che fornisce ai sistemi comunicanti le chiavi simmetriche necessarie per l'authN e/o cifratura dei pacchetti. Come distribuire le chiavi? OOB (manuale) o automatica. **Come funziona?**

1. **Negoziazione di SA-IKE bidirezionale** → viene creata una SA-IKE per proteggere l'interazione tra i 2 nodi IKE (lo scambio di chiavi avverrà tra questi 2 nodi e sarà aggiunta al SAD); con questa SA-IKE si protegge la negoziazione della SA richiesta da IPsec (quella per cui si è richiesto l'intervento di IKE) [questa SA-IKE può essere riusata per negoziare altre SA IPsec]
2. **Negoziazione della SA IPsec** (ISAKMP SA)

⚠ **VPN concentrator** = apparecchiature special-purpose che fungono da terminatori di tunnel IPsec (per accesso remoto di singoli client e per creare VPN s2s) [prestazioni >, costi <]

⚠ **Applicabilità** di IPsec → solo unicast e solo tra parti che hanno attivato una SA ("gruppi chiusi")

Altri aspetti della sicurezza in reti IP:

- **Sicurezza IP** → indirizzi non autenticati; pacchetti non protetti (integrità, authN, riservatezza, replay); dunque, attaccabili tutti i protocolli che usano IP come trasporto (soprattutto quelli di servizio [cioè non quelli di livello applicazione] [es. ICMP])
- **Sicurezza ICMP** (Internet Control and Management Protocol) → è vitale per gestire la rete (echo request/reply, destination unreachable, redirect, time exceeded for datagram...), ma attaccabile a causa della mancanza di authN.

Un attacco è lo **SMURFING ATTACK** (funziona anche con UDP), dove l'attaccante invia tanti pacchetti **ICMP echo request** ad un indirizzo broadcast usando come IP sorgente quello della vittima, facendo sì che tutti i dispositivi della rete rispondano con **echo reply** alla vittima (sommergendo la rete di traffico)

- **Sicurezza del routing** → prima c'era bassa sicurezza nell'accesso ai router per la loro gestione (si usavano password in chiaro) e nello scambio delle tabelle di routing (authN basata su IP). Un attacco è **ARP (Address Resolution Protocol) poisoning** (usato per scoprire indirizzo L2 di un nodo di cui so indirizzo L3, salvando il risultato in una ARP table), dove inserisco dati falsi in ARP table
- **Sicurezza del DHCP** → DHCP è un protocollo broadcast non autenticato che fornisce indirizzo IP, netmask, default gateway, local nameserver e local DNS suffix; è molto facile attivare un falso DHCP server e tramite questo si possono fare attacchi DoS, MITM logico e traduzione nomi-indirizzi errata
⚠ Alcuni switch offrono **DHCP snooping** (solo risposte da trusted port) e **IP guard** (solo IP ottenuti da DHCP); esiste poi *Authentication for DHCP Messages* (RFC-3118)

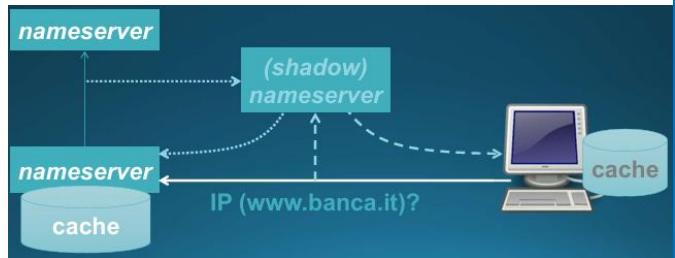
- **TCP SYN flooding** → avvengono richieste multiple con IP spoofing, saturando la tabella delle connessioni fino a che non vanno in **timeout**. Possibili **difese** sono: abbassare timeout, aumentare le dimensioni della tabella, usare un router con **SYN interceptor** (sostituisce il server nella 1^a fase; se l'handshake ha successo, trasferisce il canale al server; timeout aggressivi) usare un router come **SYN monitor** (uccide i collegamenti pendenti)

⚠ C'è anche un modo efficace per evitare del tutto il SYN flooding, ovvero il **SYN cookie** (usa il sequence number del pacchetto SYN-ACK per trasmettere il cookie al client e riconoscere così i client che hanno già inviato il SYN senza memorizzare niente sul server) [non usato però in generale]

- **DNS (Domain Name System)** → fa la traduzione da indirizzi IP a nomi/url (ricordati solita struttura gerarchica che però parte dalla cache local dell'utente [NS = *nameserver*]). Non ha però nessuna sicurezza: soluzione è **DNS-SEC** (per evitare appunto gli attacchi sotto scritti).

Attacchi possibili sono:

- **DNS shadow server** → si fa sniffing per intercettare le query + spoofing per generare risposte false (DoS o re-direzione del traffico su siti falsi):



- **DNS flash crowd** → moltissimi utenti che fanno richieste al NS (victim)
- **DNS cache poisoning** → attirare la vittima a fare una query sul proprio NS + fornire risposta anche a query non effettuate per forzare/sovrascrivere la cache della vittima; oppure fare una query e fornire subito la risposta falsa per inserirla nella cache della vittima

Dopo che Kaminsky ha scoperto un nuovo attacco che rende il cache poisoning ancora più facile, DNS-SEC è diventato fondamentale da implementare.

DNS-SEC fornisce:

- firma digitale dei record DNS (nessuna firma delle query DNS, ma solo dei dati in esse)
- nessuna root CA (ogni DNS zone ha una coppia di chiavi asimmetriche, dove le chiavi pubbliche sono firmate dal DNS parent)
- trust anchor (punto di partenza per la validazione)

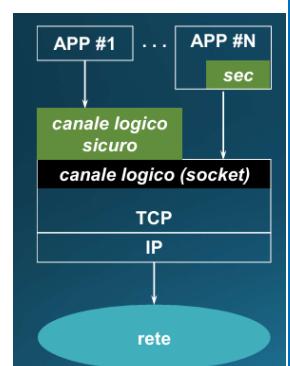
Ma ha anche dei **problemi**:

- nessuna sicurezza nel dialogo tra DNS client e DNS local server (vanno usati protocolli di protezione canale)
- crittografia sui server DNS (> sovraccarico)
- dimensione dei record
- non è automatico DNS-SEC, va abilitato dai network provider

6) SICUREZZA DI CANALE

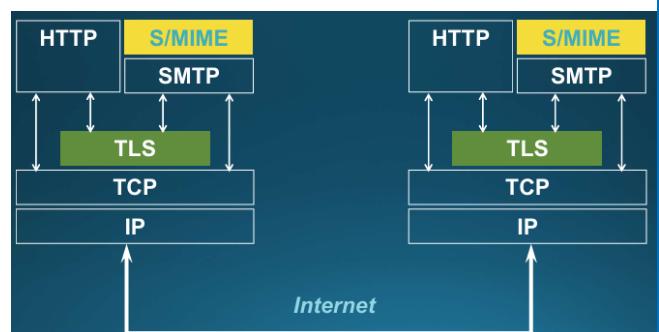
La **SICUREZZA DI CANALE** avviene mediante authN (singola o mutua), integrità e segretezza **solo durante il transito nel canale**, senza possibilità di non-ripudio + richiedendo poche modifiche alle applicazioni.

Il livello sessione sarebbe ideale per implementare molte funzioni di sicurezza, ma non esiste in TCP/IP, quindi è stato proposto un livello "**sessione sicura**" (semplifica il lavoro degli sviluppatori applicativi, evitando possibili errori di implementazione e rimanendo a scelta dell'applicazione).



Il protocollo proposto per **trasporto sicuro** (authN, riservatezza msg, integrità msg, protezione da replay e da filtering) [es. nei pagamenti] è **SSL (Secure Socket Layer)**, applicabile a HTTP, SMTP, FTP...

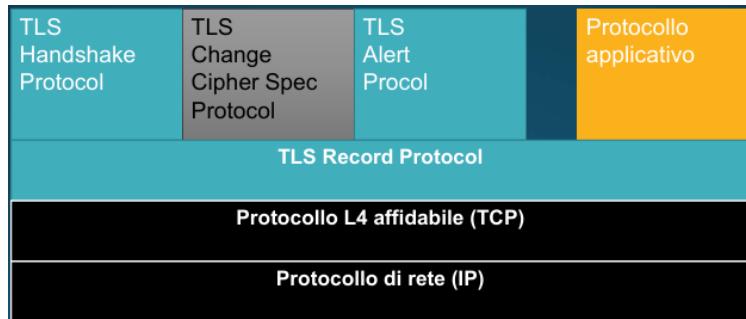
Il successore di SSL (la sua evoluzione) è **TLS (Transport Layer Security)**, il quale fa **enfasi su algoritmi crittografici e digest standard** (non proprietari); inoltre permette **coesistenza con altri protocolli** (guarda →)



TLS ha una serie di *porte ufficiali* per applicazioni TLS. Vediamo alcune **caratteristiche** di TLS:

- **apertura del canale** → server si autentica presentando la propria chiave pubblica (certificato X.509) e subendo una sfida asimmetrica (con TLS 1.2 è opzionale); l'authN del client è opzionale (solo su richiesta del server)
- **per authN + integrità dei dati scambiati**, il protocollo prevede l'uso di algoritmi simmetrici, un MAC/AE e un MID per evitare replay e cancellazione
- **scambio di chiavi** → server per authN dei dati + riservatezza. Avviene guidato dal client (anche chiamato RSA key exchange) così: client genera segreto (o detto "pre-master secret"); segreto viene comunicato al server cifrandolo con la sua chiave pubblica (RSA); chiavi scambiate con DH/ECDH

Architettura di TLS:



Cosa c'è nel TLS Record Protocol?



⚠ **TLS Handshake Protocol** = passi: richiesta HTTPS → configurazioni sicurezza → certificato (da server) → server key exchange (challenge/response) → certificato (da utente) → client key exchange (challenge/response) → **canale sicuro attivo!**

[in pratica ho scambiato: algoritmo di authN, tipi certificati, nome algoritmo di scambio chiavi, pre-master secret, nome algoritmo hash, nome algoritmo cifratura simmetrica, 2 numeri random]

Cipher suite = riassume gli algoritmi da usare per:

- **scambio di chiavi e authN**
- **cifratura simmetrica**
- **hash (per calcolo del MAC o generazione chiavi)**

TLS_DH_DSS_WITH_AES_128_CBC_SHA
TLS_DH_RSA_WITH_AES_128_GCM_SHA
TLS_DHE_DSS_WITH_AES_128_CBC_SHA
TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA256

Session-id = in una tipica transazione web, **se ogni volta si dovessero rinegoziare i parametri crittografici per SSL, il collegamento si appesantirebbe troppo**; quindi la soluzione è:

- **dopo un handshake normale** → server restituisce un *session-id*
- se all'apertura della connessione, il client presenta un *session-id* valido si **salta la negoziazione**, e si va subito avanti con la comunicazione usando algoritmi e segreti precedentemente negoziati
- server può rifiutare l'uso del *session-id*

⚠ TLS Handshake Protocol con *session-id* = passi: richiesta HTTPS → *session-id* → canale sicuro attivo!

Ogni volta che scarico un oggetto dal sito, viene scaricato con chiave diversa, quindi non riuso, quindi attaccanti non possono usarle!

Meccanismi effimeri = chiavi usa-e-getta generate al volo che devono essere firmate (es. X.509) [quindi si usano DH/ECDH, non RSA perché lento]

⚠ *Perfect forward secrecy* → senza metodi effimeri, se attaccante scopre chiave privata può decifrare tutte le sessioni; con meccanismi effimeri, la chiave privata del server viene usata solo per firmare, quindi non può decifrare le sessioni

⚠ *Heartbleed* = mediante **TLS heartbeat** (tenere una connessione aperta senza rinegoziare continuamente una sessione SSL), un attaccante può inviare una richiesta “truccata” e farsi restituire porzioni di **memoria del server** (chiavi, password, dati sensibili) che non dovrebbero essere esposte

DTLS (Datagram TLS) = applica i concetti TLS alla sicurezza dei **datagram (UDP)**, quindi offre meno sicurezza (non si appoggia su TCP, ma su UDP). Ha record esplicativi, sequence number esplicito e ridefinisce il concetto di integrità (qualche modifica ai datagram è concessa); quindi anche più difficile distinguere attacchi da problemi di rete

OpenVPN = alternativa a IPsec per creare canali sicuri e VPN, basata su TLS (ricorda da sopra: TLS fa comunicazioni client-server; è asimmetrico, si negoziano algoritmi e chiavi con handshake e si può aggiungere client authN). IPsec è a livello kernel (scomodo e meno affidabilità nelle prestazioni); però bisogna fare attenzione che il codice di OpenVPN è “open” ma sviluppato da terze parti (a pagamento infatti), quindi non affidabile quanto IPsec

SSH (Secure SHell) = pur nascendo come protocollo per eseguire comandi shell remote, offre protezione di canale, vari tipi di authN (indirizzi IP, chiave RSA di host e chiave RSA di utente); usa diverse cipher suite (focus su AES, ma anche keyed digest e HMAC) e la chiave di cifratura è una chiave di sessione scambiata in fase di connessione (DH/ECDH).

Usa **chiavi RSA, ma non certificati**; la chiave pubblica (non essendo associata ad un certificato di una CA) deve essere nota alla controparte per poter essere verificata (infatti tiene un DB con tutte le chiavi conosciute). **AuthN** funziona così: *client* = uso di login + pwd in canale cifrato o challenge RSA puro; *server* = solo challenge RSA. È leggero, ma difficile da gestire!

⚠ **Dynamic port forwarding** = tunnel SSH (che permette di implementare un proxy SSH); si apre un canale sicuro SSH verso un altro host, e tutto il traffico verso le altre destinazioni passa da questo host (e viene poi inoltrato a destinazione)

7) SICUREZZA DI MESSAGGIO (o dei DATI)

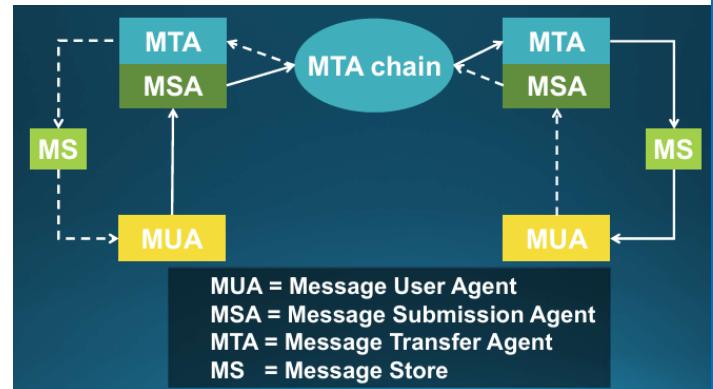
SICUREZZA DI MESSAGGIO = autenticazione (singola), integrità e segretezza contenute direttamente nel messaggio (quindi io [mittente] **proteggo direttamente i dati prima che escano sul canale**) [quindi a differenza del canale, sicurezza basta dal mittente]. **Non-ripubblio garantito** (il canale no!); **richiede modifica alle applicazioni** (ogni app implementa la sua sicurezza al suo interno, quindi è difficile garantire *interoperabilità* [la parte in comune si limita ai canali di comunicazione, ovvero i socket]).



⚠ S-HTTP = obsoleto (sicurezza di canale è sufficiente); fu inventato partendo da HTTP-1.0, ma pieno di errori e problematiche

POSTA ELETTRONICA SICURA = ci sono vari protocolli, porte e formati per le email:

- **protocolli di invio email** → **SMTP** (Simple Mail Transfer Protocol), **ESMTP** (Extended SMTP)
- **protocolli di accesso al MS** (Mail Store) → **POP** (Post Office Protocol), **IMAP** (Internet Message Access Protocol) [per garantire authN utente, authN server, riservatezza/integrità della posta]
- **formati delle email** → **RFC-822** (solo 64 caratteri disponibili), **MIME** (estensione multimediale a RFC-822)



⚠ Problematiche:

- sistema *connectionless* (store-and-forward)
- *MTA non fidati*
- sicurezza del *MS* (dove le email stanno fino a che l'utente non le scarica)
- gestire *confidenzialità* di msg anche con mailing-list
- compatibilità con *legacy software*

SMTP/ESMTP = protocollo per comunicare con MTA e inviare una mail (header + body); i client si presentano con:

- **HELO hostname** → per **SMTP**
- **EHLO hostname** → per **ESMTP** (se il server ricevente parla ESMTP, deve dichiarare le estensioni che supporta, riga per riga, nella sua risposta all'EHLO)

Esempio (SMTP + TLS):

```

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

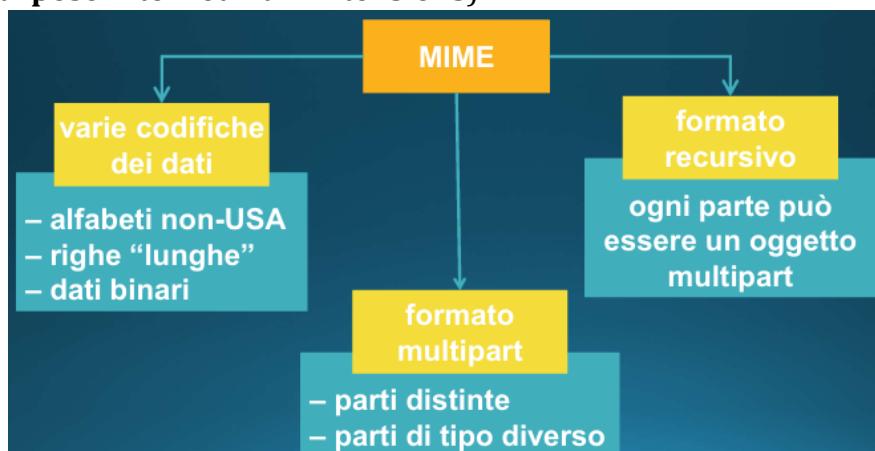
```

⚠ **RFC-2595** (**TLS per POP/IMAP**): prima si apre il canale applicativo, poi si negozia la sicurezza tramite un comando apposito (STARTTLS per IMAP, STLS per POP₃). Client e server devono poter essere configurati per rifiutare user e pwd; client confronta identità del certificato con l'identità del server

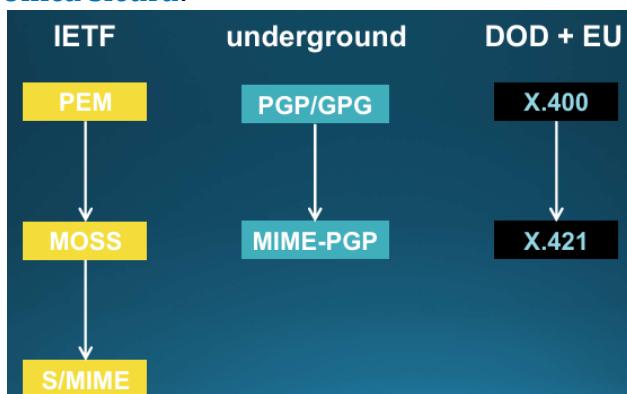
FORMATI e SICUREZZA

- **RFC-822** → solo chars *US-ASCII (7 bit)*; righe terminate da **<CR><LF>**; msg composti da **header + body**, dove:
 - header = parole chiave a inizio riga; righe di continuazione iniziano con uno spazio; mittente e destinatario possono essere diversi dai nomi utente
 - body = separato dall'header da una riga vuota; contiene il msg
- **MIME (Multipurpose Internet Mail Extensions)**

• From:	mittente (logico)
Sender:	mittente (operativo)
• Organization:	organizzazione del mittente
• To:	destinatario
• Subject:	argomento
• Date:	data e ora di spedizione
• Received:	passaggi intermedi
• Message-Id:	ID di spedizione
• CC:	in copia a
Bcc:	in copia (nascosta) a
• Return-Receipt-To:	ricevuta di ritorno a



- Formati di posta elettronica sicura:



⚠ Proprietà di sicurezza per msg e-mail:

- **integrità** (msg non modificabile)
- **authN** (identifica mittente)
- **non-ripudio** (mittente non può negare di aver spedito la mail)
- **riservatezza** (opzionale) [sia in transito sia nella casella di posta]
- **funzionalità** (nessuna modifica agli MTA, minime modifiche ai MUAs)

⚠ Sicurezza delle email:

- ❖ **algoritmi simmetrici** (crittografia dei msg)
- ❖ **algoritmi asimmetrici** (per scambiare la chiave simmetrica, per la firma digitale)
- ❖ usare **certificati a chiave pubblica** [es. X.509] (anche per non-ripudio)
- ❖ sicurezza del msg si basa **solo sulla sicurezza dell'UA del destinatario**, non su quella degli MTA (non fidati)

S/MIME = per proteggere messaggi MIME:

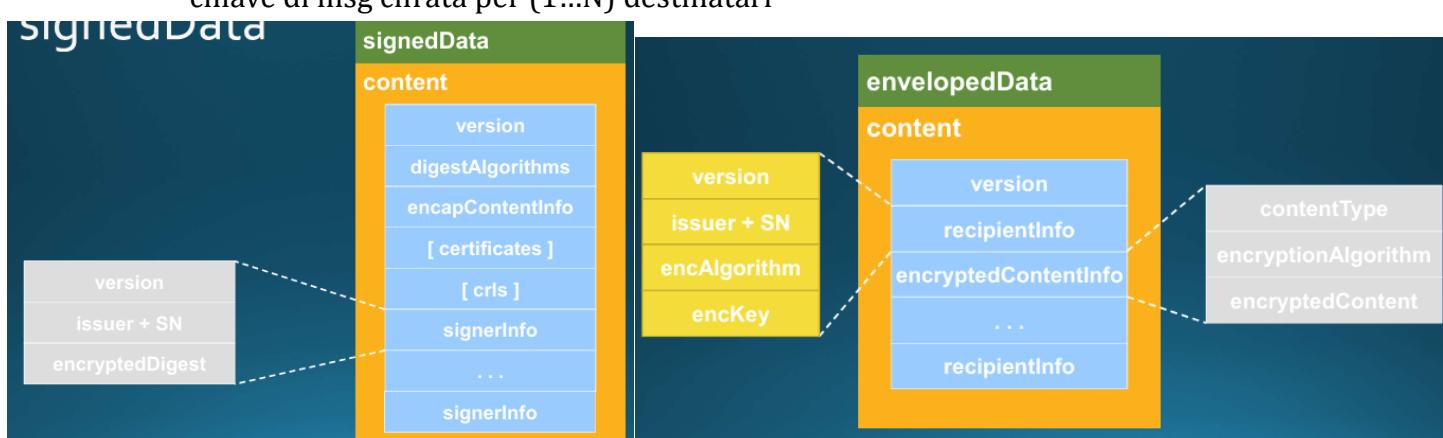
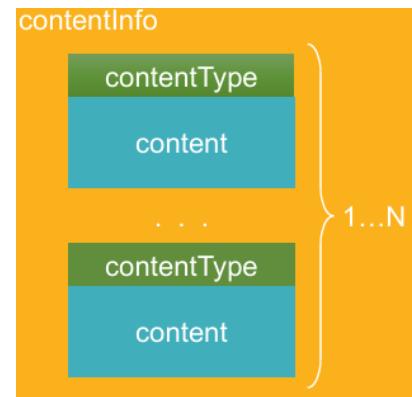


- **PKCS#7 (S/MIME v2)** e **CMS (S/MIME v3)** → formati crittografici standard per “*buste sicure*”. CMS è l’evoluzione di PKCS#7; supporta authN, integrità e confidenzialità dei dati con algoritmi simmetrici o asimmetrici. Supporta **firme multiple** sullo stesso oggetto e **formato ricorsivo**. Tipi di msg sicuri:

- *clear-signed* → msg in chiaro + firma; solo chi ha MUA sicuro può verificare la firma
- *signed* → msg + firma codificati; solo chi ha MUA sicuro può decodificarli e verificare la firma
- *encrypted/enveloped* → msg cifrato + chiavi cifrate, codificato; solo chi ha MUA sicuro (e le chiavi) può decifrarlo
- *signed/enveloped*

Altre operazioni sono **canonicalizzazione** (rendere standard il formato crittografico, indipendente da OS/host/net) e **codifica** (evitare alterazioni non-malevole da parte degli MTA)

- **CMS** → struttura:
 - **data** = semplici sequenze di byte
 - **signedData** = dati + (1...N) firme parallele sequenziali
 - **envelopedData** = dati cifrati con crittografia simmetrica + chiave di msg cifrata per (1...N) destinatari



- **X.509** → formati dei **certificati a chiave pubblica** per firma digitale/cifratura

⚠ **PKCS#12** (security bag) → struttura dati per il trasporto di materiale crittografico; trasporta l’**id digitale** di un utente

MAIL SPAMMING = invio di messaggi indesiderati (pubblicità, malware, phishing); è la maggior parte delle email oggi. Strategie degli spammer sono:

- **nascondere vero mittente**
- **spedire spam tramite MTA speciali** (open mail relay, zombie/botnet o indirizzi IP variabili/inesistenti)
- **content obfuscation + superamento controlli anti-spam**

Anti-spam per:

- **outgoing MSA**
 - non configurare i propri MSA/MTA come “open relay”, ma restringerne solo ai propri utenti
 - autenticare gli utenti dei propri MSA
- **incoming MTA**
 - BlackList o WhiteList per rifiutare o accettare mail da un MTA
 - URI-based (DNS-based BlackList)
 - time-based (GreyListing) [try later + OK se si ripresenta lo stesso MTA dopo il tempo richiesto]
 - nolisting

8) FIREWALL (FW) e IDS

FIREWALL (in America case di legno, quindi serve un “*muro tagliafuoco*” di materiale non-infiammabile per non propagare l’incendio del vicino) = collegamento controllato tra reti a diverso livello di sicurezza (*sicurezza del perimetro [filtro di rete]*) [**ingress** firewall = autorizzazione per collegamenti in ingresso, **egress** firewall = autorizzazione per collegamenti in uscita; **distinzione facile per servizi orientati al canale [TCP]**]

⚠ Firewall **non si compra, si progetta**, quindi va trovato trade-off tra sicurezza, funzionalità e costo

3 principi dei FW:

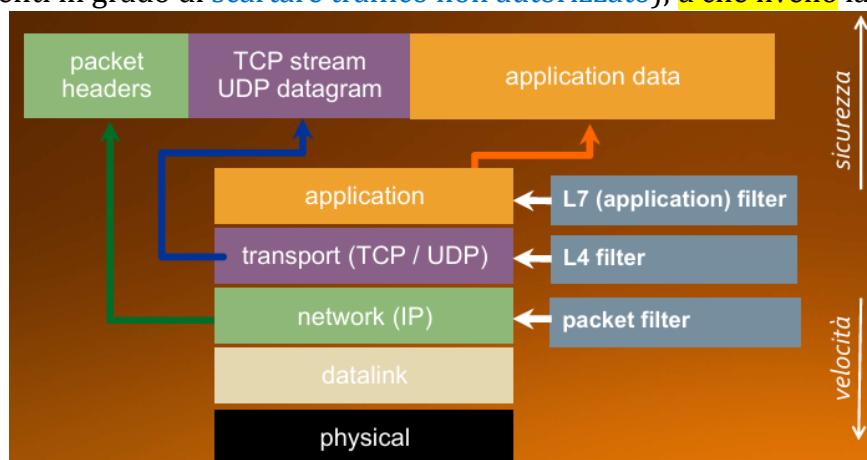
1. FW deve essere l’**unico punto di contatto** della rete interna con quella esterna
2. **solo traffico autorizzato** può attraversare FW
3. FW deve essere **altamente sicuro**

Politiche di autorizzazione:

- **WhiteListing** → tutto ciò che *non è espressamente permesso*, è vietato (> sicurezza, > difficoltà)
- **BlackListing** → tutto ciò che *non è espressamente vietato*, è permesso (< sicurezza, < difficoltà)

Elementi di base di un FW:

- **Filtri** (componenti in grado di **scartare traffico non autorizzato**); a che livello lavorano?



Si usano varie tecnologie a seconda dei vari livelli di rete:

- **PACKET FILTER (STATIC)** → controlla **singoli pacchetti IP** (con dati presi da IP header, transport header, 5-tuple di IP), non considera le connessioni. È disponibile sui router (*screening router*)
 - Pro:** indipendente dalle applicazioni, ottime prestazioni, basso costo
 - Contro:** difficile supportare servizi con porte allocate dinamicamente, difficile authN di utenti

- **STATEFUL PACKET FILTER (DYNAMIC)** → simile a packet filter, ma “*state-aware*”. Prende informazioni di stato dal livello trasporto e/o applicativo. **Distingue le nuove connessioni da quelle già aperte** (mantiene tabelle di stato per le connessioni aperte)
 - Pro:** prestazioni simili a packet filter (connessioni già autorizzate non richiedono altri controlli)
 - Contro:** limitazioni (problemi con utenti, protocolli di alto livello...)
- **APPLICATION-LEVEL (L7 FILTER)** filter → **ogni app richiede uno specifico componente**; interviene dopo che il canale è stato decifrato (content/packet inspection + lawful interception)
 - Contro:** ritardo nel supporto delle nuove app, consumo risorse, < prestazioni
- **Gateway** [= *proxy*] (**svolge il lavoro per conto di un altro componente**; implementa politica di autorizzazione per decidere quale lavoro svolgere)
 - **CIRCUIT-LEVEL gateway** → è un proxy **non “application-aware”** (lavora su L4/L5); crea circuito tra client e server a livello trasporto (non ha però nessuna comprensione dei dati in transito).
 - Riassembra i segmenti TCP o datagram UDP** (se rispettano regole di autorizzazione, li inoltra su un'altra interfaccia di rete) [riassempolare pacchetti IP protegge da alcuni attacchi L3/L4].
 - Rompe il modello C/S** per una specifica connessione (server più protetti + può autenticare i client), ma rimangono molte limitazioni del packet filter
 - **APPLICATION-LEVEL gateway** → esamina il contenuto dei pacchetti a livello applicativo (L7); implementano direzionalità e **rompe anch'esso il modello C/S** (/// + mancanza di trasparenza per i client). Vediamo alcuni tipi:
 - **HTTP (forward) proxy** = server HTTP che fa solo da front-end e passa le richieste dei client al vero server (**esterno**); quindi si colloca tra rete interna e rete esterna. Fa il caching delle pagine esterne per tutti gli utenti interni; fa authN + authZ degli utenti interni
 - **HTTP reverse proxy** = server HTTP che fa solo da front-end e passa le richieste al vero server (**interno**) [va nella direzione opposta del forward; aiuta i servizi che vogliamo erogare verso l'esterno]; ha servizi aggiuntivi come obfuscation, acceleratore SSL e web, load balancer, compressione...
 - **WAF** (Web Application Firewall) = **filtrare traffico L7**; conosce nativamente HTTP (comandi, header, contenuto) [ModSecurity → nasce come WAF, ma usato anche come L7 filter]

Su L2 abbiamo **STEALTH FIREWALL** (un firewall **privo di indirizzo di rete** [IP, L3] quindi non attaccabile direttamente; intercetta i pacchetti fisicamente [basandosi su propria policy] e li scarta/copia, ma non li altera).

Infine abbiamo **LOCAL/PERSONAL FIREWALL** (installato direttamente sul nodo, quindi **può accedere direttamente ai processi dell'OS** che decide quali operazioni sono disponibili da interno/esterno; potendo accedere direttamente ai processi, può filtrare in base al nome utente, al PID etc...)

Possibili **ARCHITETTURE di firewall**:

- **SCREENING ROUTER** → **usa il router per filtrare il traffico sia a livello IP (L3) sia superiore**; non richiede hw dedicato e non necessita di proxy e modifiche agli applicativi

Pro: facile, economico

Contro: insicuro

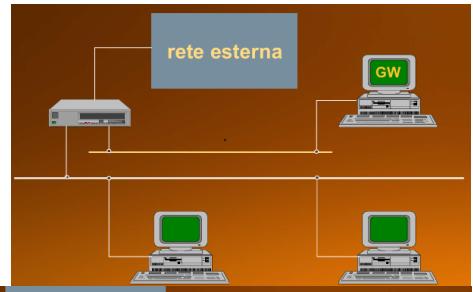
- **DUAL-HOMED GATEWAY**

Pro: facile, richiede poco hw, possibile mascherare rete interna

Contro: poco flessibile, grosso sovraccarico sul GW



- **SCREENED HOST GATEWAY** → non tutto deve passare per forza dal GW: il router blocca traffico da interno (tranne se arriva dal bastion) e traffico da esterno (tranne se verso il bastion)
 - Pro:** flessibile
 - Contro:** costoso, complesso, si possono mascherare solo gli host/protocolli che passano dal bastion

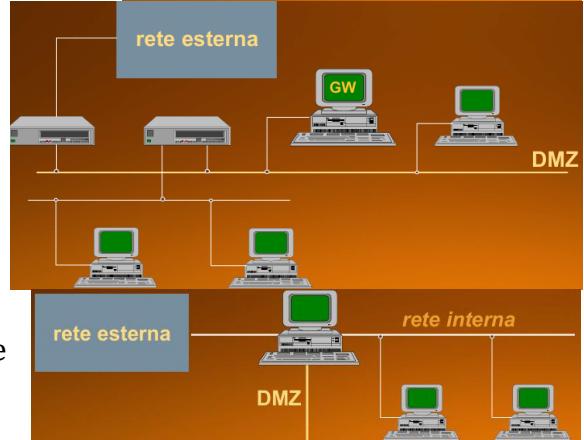


- **SCREENED SUBNET**

Pro: sulla DMZ (Zona De-Militarizzata), oltre al gateway, ci possono essere più host (tipicamente i server pubblici); si può configurare il routing in modo che la rete interna sia sconosciuta

Contro: costoso

⚠ Architettura screened subnet **versione 2**: per motivi di costo e complessità, spesso si omettono i router (e le loro funzioni sono nel GW) [“firewall a 3 gambe”]



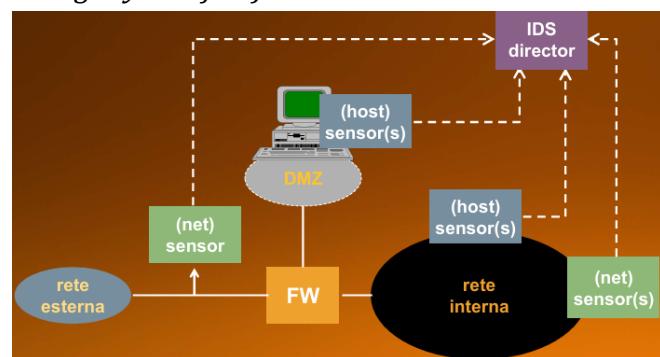
⚠ Dunque i firewall sono efficaci solo contro gli attacchi su canali che sono bloccati; per gli altri canali, occorrono altre difese (VPN, IDS...) ↓

IDS (Intrusion Detection System) = identificano chi usa un computer/rete senza authZ (o senza i privilegi) sulla base di “pattern comportamentali” degli utenti senza authZ (che è diverso da quello degli utenti autorizzati) in modo da prevedere anche attacchi imminenti. Tipologia di IDS:

- **PASSIVI** → usano checksum crittografiche e fanno riconoscimento di pattern (*attack signature*)
- **ATTIVI** → fanno learning (analisi statistica del funzionamento del sistema) + monitoring (analisi attiva di traffico dati, sequenze e azioni) + reaction (confronto con parametri statistici)

2 tipi di IDS:

- **HIDS (host-based IDS)** → fa analisi dei log (dell'OS e delle app, tramite *log file monitor*), attivazione di strumenti di monitoraggio interni all'OS (*system integrity verifier*) e notifica a strumenti che raccolgono i dati da diversi sistemi
- **NIDS (network-based IDS)** → monitorano il traffico di rete; composto da sensori (controlla il traffico e log, individuando pattern sospetti), director/correlatore (coordina i sensori e gestisce il security db) e IDS message system (consente la comunicazione sicura tra i componenti dell'IDS)



IPS (Intrusion Prevention System) = per velocizzare ed automatizzare la risposta alle intrusioni; non ben visto perché c'è il pericolo di prendere la decisione sbagliata o bloccare traffico legittimo

NGFW (Next-Generation Firewall) = identifica le applicazioni e gli utenti, definisce policy per utente e per applicazione, e filtra in base a vulnerabilità, minacce e malware

UTM (Unified Threat Management) = integrazione di più prodotti in 1 dispositivo (firewall, VPN, anti-malware, IDPS...) con l'obiettivo di ridurre il n° di sistemi diversi, la complessità di gestione e il costo

⚠ **Honey pot** (o *honey net*) = specchio per le allodole per attirare gli attaccanti e fregarli/arrestarli

INCIDENT RESPONSE

- **incidente** = evento che compromette l'integrità, la confidenzialità o la disponibilità di un asset
- **data breach** = incidente che genera la rivelazione (disclosure) o la potenziale exposure di dati
- **data disclosure** = data breach per cui è stato confermato che dei dati sono stati effettivamente rivelati (non solo esposti) a persone non autorizzate

Il principio è che “**gli incidenti informatici avverranno sicuramente**” quindi bisogna essere pronti ad affrontarli (minimizzare tempi di risposta, ridurre gli errori di gestione, ridurre la probabilità che ricapiti), istituendo l'**Incident Response Team** (team dedicato, con ruoli definiti e ben gerarchizzata). Bisogna quindi **definire le politiche di risposta** agli incidenti e piani di azione, e preparare le linee guida che definiscono le **priorità** (quali incidenti affrontare prima), ma bisogna anche **identificare gli incidenti il prima possibile** (*monitoraggio* grazie a logging e IDS e analisi automatiche di tanti dati).

Bisogna anche preparare **documentazione** relativa alla comunicazione durante gli incidenti e **imparare dagli incidenti precedenti**.

Vediamo alcuni strumenti per gestire questi incidenti:

- **SIEM** → combina 2 funzionalità: **SEM** (Security Event Management [monitoring su log]) e **SIM** (Security Information Management [analizza e produce report su dati di log]); disponibile come sw, appliance o in cloud. Permette di osservare e registrare le attività di un sistema informativo
- **EDR (Endpoint Detection and Response)** → combina real-time continuous monitoring, aggregando i dati dei nodi di un sistema informativo e permettendo di specificare *regole di reazione automatiche*
- **SOC (Security Operation Center)** → **dipartimento che gestisce tutte le funzionalità di sicurezza**, per agevolare la comunicazione, la gestione e la prevenzione degli incidenti. Include i responsabili degli asset da proteggere (locale e cloud) e le **misure preventive** (preparazione del personale, trend del cybercrime, protezione degli asset, vulnerability management & patch management). Fa quindi continuous proactive monitoring del sistema (**integra SIEM e EDR**) e risponde agli attacchi per minimizzare gli effetti. Inoltre, si occupa anche di **compliance management**

9) STANDARD DI SICUREZZA INFORMATICA

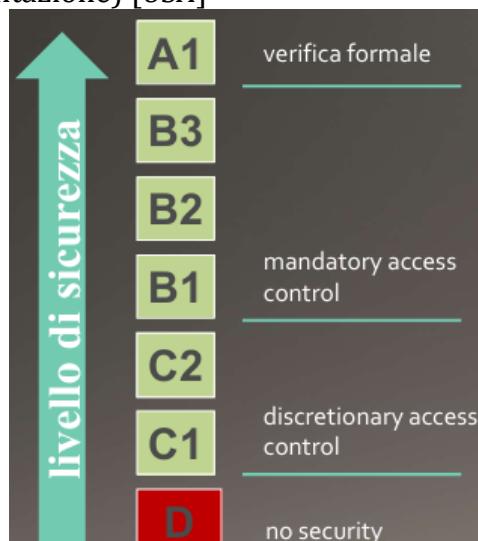
Gli **STANDARD DI SICUREZZA** aiutano a scegliere prodotti e aziende che rispettino certi requisiti di sicurezza, tramite una verifica fatta da qualcun altro (un tempo c'erano i “*tiger team*” che provavano a rompere la sicurezza singolarmente).

Si è quindi creato un **ECOSISTEMA di cybersecurity basato su standard**. Gli standard di sicurezza servono anche come **certificazione** richiesta per l'accesso a certi mercati; queste certificazioni hanno il processo di valutazione ben spiegato e anche il certificato stesso è ben spiegato

⚠ La sicurezza viene quindi valutata, ma sempre fino ad un certo livello, quindi non “ci assicura” la completa sicurezza, ma ci dice solo che una certa versione del prodotto ha superato un certo tipo di test in determinate condizioni

La *prima certificazione standard* fu ORANGE-BOOK (del dipartimento della difesa americana)

CLASSIFICAZIONE SULLA BASE DI REQUISITI (politica di sicurezza, controllo delle responsabilità, garanzie di affidabilità e documentazione) [USA]



In Europa, c'era la certificazione **ITSEC/ITSEM**, che valutava sistemi di vario tipo (**target of evaluation [TOE]**). Qui a differenza degli USA, il processo di valutazione era descritto esplicitamente (non veniva fatto "a sorpresa") e c'era possibilità di upgrade. ITSEC si concentra su **efficacia** e **correttezza** e non è una valutazione **assoluta** (si poteva chiedere una valutazione direttamente su un certo livello) [se cambiava la TOE, andava fatta la **rivalutazione**]

Dopo che gli USA hanno introdotto **FC-PIPS** (certificazione per prodotti informatici), si è deciso di creare una **certificazione unica nel mondo**, ovvero **Common Criteria (CC)**: sono **criteri** (quindi non fanno riferimento a processi di sviluppo, metodi di valutazione e regolamentazioni/leggi).

Quindi i **common criteria** sono gli stessi ovunque nel mondo, ma ogni stato definisce il suo **evaluation scheme** (per allinearsi alle regolamentazioni locali, quindi ci sono tante piccole differenze).

⚠ **Protection Profile (PP)** = specifica formale dei requisiti di sicurezza indipendente dal TOE ("per le categorie standard vi diciamo direttamente come rispettare i common criteria")

Quindi abbiamo degli **standard relativi ai prodotti**: **BS-7799 (ISO-17799)** → metodologia di sviluppo, rilascio, controllo e audit di un sistema di sicurezza (*best practice*); è stata la base di **ISO-27000** (insieme di raccomandazioni su come gestire la sicurezza delle informazioni a livello aziendale [con focus su gestione dei rischi, **ISMS (Information Security Management System)**]) **[ISO-27001]** definisce come creare un ISMS; richiede analisi dei rischi e le decisioni si basano su questa analisi]

⚠ **Altri standard specifici di settore:**

- **PCI DSS (Payment Card Industry Data Security Standard)** → standard che prevede certificazione per l'uso della carta di credito sul loro sito/piattaforma (**REQUISITI**: secure networks/systems, protect account data, vulnerability management program, access control measures, monitor & test networks, information security policy)
- **ISO-21434** → cybersecurity per **automotive** + valutazione del rischio (tutti i veicoli devono averlo)

GDPR & Cybersecurity Act

GDPR (General Data Protection Regulations) è fatto da Regulation (articoli) e Recitals (spiegazioni nel corpo del GDPR) ed è focalizzato sulla **"privacy"** dell'utente; tratta **ogni informazione che permetta di identificare una persona** (DB aziendale, registrazioni di sorveglianza, foto, e-mail con dati sul mittente, dati di monitoring...) e riguarda **sia sistemi** di acquisizione di dati **automatici sia manuali**.

I **dati sensibili** (*special category personal data*) sono razza, opinioni politiche, religione, sindacati, dati biometrici/genetici, sulla salute, orientamento sessuale; in particolare i dati giudiziari (*fedina penale*).

Ruoli:

- **Trattamento** (*data processing*) → qualunque attività che tratta dati personali; secondo il GDPR, i dati devono essere **processati secondo la legge, in maniera onesta e trasparente**. I trattamenti devono essere “**minimi**” (solo i dati necessari e dichiarati), “**precisi**”, **non-memorizzati** più del tempo necessario agli scopi dichiarati e i sistemi che usano dati personali devono avere **Privacy by Design (PbD)** [ovvero “**proactive not reactive**” + “**privacy by default**” (max protezione) + “**privacy embedded into design**” + “**full functionality**” (non limitare le funzionalità del sito a chi vuole il rispetto della propria privacy) + “**full lifecycle protection**” + “**visibility and transparency**” (ovvero mi fido ma qualcuno deve verificare → trust but verify) + “**respect for user privacy**”]
- **Responsabile** del trattamento (*data controller*) → definisce come e con che scopo i dati personali siano processati (gli scopi devono essere **legittimi, esplicativi e dichiarati**); secondo il GDPR, il responsabile deve essere in grado di fare accountability (dimostrare che sta facendo le cose in modo appropriato) e reporting
- **Incaricato** (*data processor*) → processa i dati (supervisionato dal controller)
- **Supervisory Authority** → entità nazionale che controlla la gestione del GDPR in uno stato (*garante*)

⚠ **PIA** (Privacy Impact Assessment) = come analisi dei rischi, ma richiesta esplicitamente dal GDPR

⚠ **DPO** (Data Protection Officer) = nuovo ruolo obbligatorio introdotto dal GDPR; può essere interno o esterno all'azienda (in pratica è il responsabile dei dati [comunica data breach al Garante]); paga lui i danni se le cose vengono fatte male

Se avviene **data breach**, va notificato alla Supervisory Authority (in Italia, il *Garante*) ma anche agli utenti interessati (**personal data breach** = alterazione e diffusione non autorizzata dei dati personali). In UE (nel *cyberspazio* della UE), i dati personali sono più protetti (allineamento nella gestione dei dati personali), ma fuori da qui bisogna affidarsi agli accordi di condivisione dei dati.

Altre leggi e enti:

- **Cybersecurity Act** → strategia dell'UE per la sicurezza cibernetica: comportarsi in modo **allineato contro le minacce informatiche** + creare **mercato unico della cybersecurity** (da certificazioni nazionali a **certificazione europea**)
- **ENISA** (aiuta gli stati membri dell'UE per la gestione operativa degli incidenti informatici) e **ECSO** (tecnologia e ricerca nell'ecosistema europeo della cybersecurity)
- **NIS2** e **DORA** → regolamenti per la protezione di sistemi informativi e delle infrastrutture critiche (**NIS2** = rafforzare requisiti di cybersecurity per organizzazioni che forniscono servizi essenziali o importanti; **DORA** = garantire la continuità dei servizi finanziari [banche, assicurazioni...] anche in caso di incidenti cyber, riducendo il rischio derivante da ICT)
- **ACN** (Agenzia Cybersecurity Nazionale)
- **CRA (Cyber Resilience Act)** → introduce **requisiti di cybersecurity obbligatori per i prodotti** con elementi digitali per tutto il ciclo di vita (in pratica cybersecurity diventa requisito di conformità di prodotto per poter vendere in UE [quindi gestisce vulnerabilità sw e hw])
- **Cyber Solidarity Act** → rafforza la capacità collettiva di prevenzione, rilevamento e risposta agli incidenti informatici su larga scala (**coinvolgendo tutta la UE**, non il singolo stato lasciato solo)