

Cyber Physical Security

Introduzione

La componente umana è da includere in ogni design di infrastruttura che andremo a disegnare. es. il problema degli utenti è che usano password deboli, scaricano gli .exe nelle mail, espongono i loro dati in maniera compromettente.

Che tecnologie sono coinvolte in un attacco informatico: sistemi, reti e programmi

Perché si effettuano tali attacchi: cattura di informazioni, soldi o creazione di disordini sociali a scopo di lucro, politico, ... a volte anche etici, ma sono molto rari adesso.

Le 3 caratteristiche fondamentali di come compromettere un sistema informatico sono queste:

- **Confidenzialità** compromessa (dell'informazione che gestisce) => accessing
- **Integrità** => changing
- **Disponibilità** => interrupting

In generale parliamo di compromettere la *confidenzialità* in quanto a dati sensibili. Sentiremo parlare del fatto che un attaccante riesce ad accedere a dati sensibili per cui non aveva autorizzazione: qualcuno mi ruba il cellulare e ci entra dentro; qualcuno è in grado di installare un keylogger che estrae informazioni dal mio cellulare. Un attacco di questo tipo, alla confidenzialità è lo *sniffing*. Se violo la confidenzialità su un db si parla di *data breaches*.

Quando parliamo di *integrità* possiamo parlare di integrità dei dati, ma possiamo anche parlare di integrità del servizio. Qualcuno non è interessato a leggere i dati quanto a modificarli, es. studenti a scuola violano il registro elettronico e cambiano i propri voti. Non vogliono violare la confidenzialità, ma vogliono modificarli e la sicurezza informatica è violata in termini di cambiamento di dati persistenti. Voglio compromettere un'industria rivale e gli faccio un attacco al fine che il sistema non funzioni più allo stesso modo (comprometto l'integrità dei dati e del servizio).

Quando parliamo di compromettere la disponibilità c'è il denial of service (DOS), impedisco ad altri di accedere al servizio.

In inglese *security* e *safety* hanno due significati diversi:

- security, sicurezza informatica
- safety, sicurezza fisica

Quando progettiamo il sistema informatico collegato a quello fisico dovremmo preoccuparci della safety!

Si ribaltano così alcune delle priorità. Di base per noi la cosa più importante è la NON compromissione della confidenzialità, ma per il mondo industriale non è così importante, è più importante la safety. Per garantire la safety, le due caratteristiche fondamentali sono l'integrità e la disponibilità. Questa non deve mai essere messa in pericolo.

Nella parte di sicurezza cyber physical cosa andremo a vedere concretamente?

- Industrial Environments: ambito industriale già citato.
- Automotive Environments: abbiamo delle automobili che hanno un sistema di infotainment, centralina, ecc adas e via dicendo.
- IoT environments: qui il problema non è tanto avere apparecchiature informatiche collegate a sistemi critici (auto, macchine industriali) quanto uno studio differente del progetto di sicurezza stesso. Dobbiamo pensare che abbiamo in modo obliquo sensori di ogni genere e sorta. Queste caratteristiche ci fanno pensare a sicurezze differenti perché gli attaccanti hanno un territorio di attacco molto diverso.

Nell'ambito del software se ho problemi rilascio semplicemente una patch, negli ambiti physical può essere complicato e nell'ambito industriale questo non è assolutamente accettabile in quanto aggiornare un sistema non è banale. Infatti si prediligono sistemi consolidati e ampiamente collaudati.

Concetti di crittografia

La crittografia è una branca della scienza che studia come trasformare i messaggi in modo da proteggere le informazioni. Va distinta la crittografia dalla *steganografia* che non mira a trasformare i messaggi per proteggerli ma a nasconderli all'interno di altri messaggi.

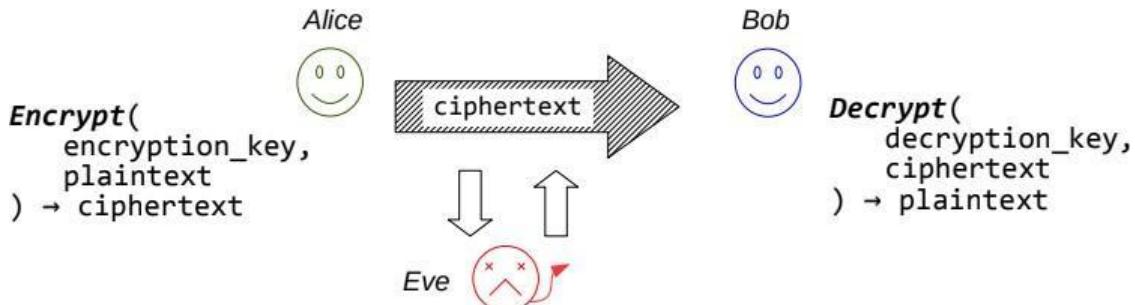
Crittoanalisi invece serve per rompere le tecniche di crittografia per ricavare le informazioni e recuperarne le originali.

La crittografia è nata in tempi molto remoti, anche se al tempo gli unici attori che avevano interesse a scambiare informazioni importanti erano i regnanti per dire o in termini militari con persone lontane ovviamente. Dal 1700 in poi la crittografia diventa più diffusa in quanto molte persone vogliono mandare informazioni su mezzi che sono accessibili ad intermediari non esattamente bene intenzionati (soprattutto per le informazioni radio).

Data at rest: dati a riposo su qualche sistema di memorizzazione

Data in sue: ci riferiamo a dati che vengono elaborati dal calcolatori (in cache per dire etc)

Ci sono degli standard diversi per i rest ma per quelli in sue ci sono delle ricerche in corso per evitare di decifrare i dati anche quando li utilizziamo.



Questo setting criptografico lo possiamo dividere in due parti:

- Setting simmetrico: ci riferiamo allo scenario precedente di comunicazione per cui le chiavi di Bob e Alice hanno due chiavi uguali. Posso chiamare la chiave a questo punto solamente *secret key*.
- Setting asimmetrico: chiave di cifratura diversa da quella di decifratura, rispettivamente *public key* e *secret key*.

Nel mondo reale vedremo come distribuire le chiavi con un setting asimmetrico è nettamente più semplice del simmetrico. Quindi a seconda del contesto opereremo di conseguenza , di modello in modello.

Crittografia simmetrica

L'obiettivo è manipolare i dati per impedire agli attaccanti di ottenere informazioni dai nostri dati.

Cifrario a trasposizione: trasformare le informazioni manipolando e riordinando simboli già presenti. I simboli utilizzati nel testo cifrato sono gli stessi di quello in chiaro ma cambiano il loro ordine. Il livello di sicurezza reale è molto basso.

Cifrario a sostituzione: un esempio noto è il cifrario di Cesare comunemente detto shift cipher. Ogni lettera di una parola si trasforma in una lettera corrispondente. Come cifrario è forte o debole? Per trovare la parola originale posso semplicemente fare un attacco brute force: tentare tutte le chiavi ammissibili in questo sistema di cifratura, per il cifrario i tentativi sono le lettere dell'alfabeto.

Cifrario a sostituzione ideale non si basa su una sostituzione data su una operazione matematica, ma su una permutazione random: essenzialmente un mapping fra un simbolo dell'alfabeto ed un altro simbolo. La chiave qua è il mapping. Dobbiamo

salvarci tutto quanto essenzialmente. Se ragioniamo su quante sono le lettere ammissibili possiamo immaginare che il keyspace è 26! Fino al 2010 sarebbe un livello di sicurezza accettabile.

Il limite di questi schemi è che sono *deterministici*: ogni lettera in chiaro viene sempre trasformata dalla stessa lettera cifrata. Tramite una distribuzione di frequenza dei caratteri posso bucare lo schema, se ho il testo in chiaro in una certa lingua posso ricavare un istogramma delle frequenze delle lettere a prescindere dalle lettere purché rimanendo nello stesso idioma.

L'unico cifrario storico pluri alfabetico è quello di **Vigenère** in cui uso una parola segreta. Ogni lettera del testo in chiaro viene shiftata tramite la parola che sto usando come chiave. La chiave ha una certa lunghezza e quando termine la chiave ricomincio di nuovo da zero (plaintext: helloworld, key: secret):

inserire esempio

Questo cifrario aumenta la resistenza ma non è infallibile. Infatti se ho abbastanza dati mi basta prendere un testo cifrato sotto campionario e riottengo il cifrario a shift originale. Se prendo una lettera ogni n dove n è la lunghezza della chiave sono sicuro che ho di nuovo una sostituzione deterministica.

I cifrari a sostituzione ideale polialfabetici sono una soluzione più avanzata, costruisco il mapping per gruppi di lettere, questo aumenta moltissimo la resistenza all'analisi di frequenza. Serve moltissimo testo per provare a rompere questa cifratura. Il difetto? Mi devo ricordare TUTTO il mapping: tutte le possibili combinazioni di sostituzione che aumentano esponenzialmente con la lunghezza del numero di lettere che considero nella sostituzione stessa.

Quando vogliamo cifrare i dati dobbiamo considerare anche termini di performance e usabilità, se ho chiavi troppo grosse ho uno schema NON pratico.

Claude Shannon nel 1949 ha notato che usare cifrari di sostituzione ideali da soli, non è sufficiente, ma se costruisco cifrari che sono degli ibridi (trasposizione + sostituzione) posso ottenere dei product cipher che sono molto più sicuri.

Kerckhoffs principle(requisiti minimi di uno schema):

- Gli algoritmi sono pubblici
- La sicurezza si basa sulla segretezza della chiave
- Computational security: Il cifrario non deve essere perfettamente sicuro, basta sapere che per romperlo ci vogliono troppe risorse a livello computazionale e di tempo. Il tipo di garanzia che vogliamo dare è che il testo cifrato deve essere per gli attaccanti indistinguibile da dati random (dati senza informazione).

Perfect Secrecy: Principio che si contrappone alla computational security, a prescindere dalla forza dell'attaccante lo schema di cifratura non può essere rotto. Questo ovviamente è teoria pura, i costi sono tali per cui non è assolutamente fattibile. Noi non la useremo mai essenzialmente. **One time pad** viene chiamato in acronimo OTP(non confonderti con One Time Password) .

L'operazione alla base della crittografia simmetrica è lo XOR. Questo ha una caratteristica fondamentale: se immaginiamo di avere una tabella di verità e un messaggio m, una chiave k e otteniamo c, immaginando di avere una chiave random, sappiamo che al 50% il messaggio in chiaro sia 0 o 1. Così come per la probabilità del messaggio in chiaro. Lo XOR è un'operazione fondamentale e ci da questa proprietà essenziale.

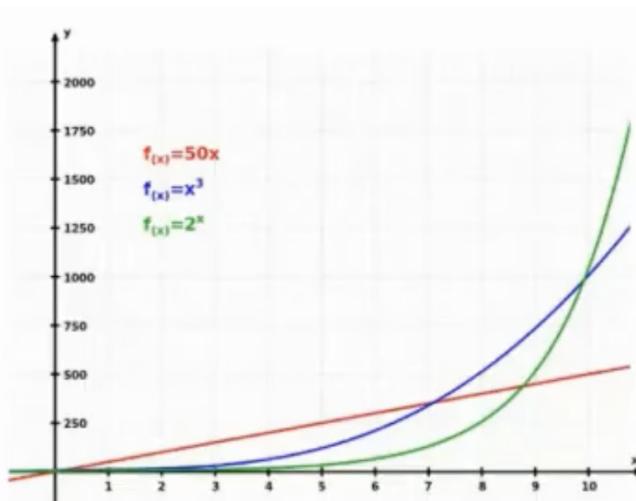
Quando cifriamo i dati e parliamo di *chiave crittografica* è essenziale che questa sia una sequenza di bit random. Fin da Cesare hai visto come ci sono state delle intuizioni geniali, come l'idea della somma modulare. Alla fine, lo XOR è una somma modulare il cui modulo è in base 2.

One time pad impone che la chiave deve essere random e lunga tanto quanto il messaggio in chiaro, questo comporta ad una completa sicurezza dello schema ma è estremamente vulnerabile per quel che riguarda l'integrità (*malleable*). Chi riceve i dati non se ne accorge della manipolazione, inoltre se l'attaccante modifica il primo bit del testo cifrato ha la certezza di star modificando il primo bit del testo in chiaro.

Computational Security

Soltanamente in crittografia posso fare delle stime concrete in termini di probabilità. Si usa però a volte un concetto algoritmico e teorico più che matematico-statistico. Cerchiamo di avere uno schema che offre 3 funzioni:

- 2 funzione efficienti: cifratura e decifratura, devono essere eseguite con poche risorse.
- 1 funzione inefficiente: l'algoritmo di rottura dello schema.



Chi cifra e decifra usando la chiave ha un costo che diverge rispetto a quello di chi attacca. Chi rompe la chiave deve eseguire una funzione esponenziale. Più aumenta la lunghezza della chiave più è

difficile per gli attaccanti in termini di operazioni.

Lo sviluppo tecnologico riduce i tempi e bisogna stare attenti perché certi schemi diventano insicuri. Come fare? Aumento la dimensione della chiave di volta in volta.

L'obiettivo è avere la funzione di attacco con valore superpolinomiale, così da avere il costo di criptazione/decriptazione molto minore rispetto a quello di 'rottura'.

Nella crittografia simmetrica il numero di bit utilizzati per la chiave corrisponde circa a 2^N operazioni per rompere lo schema.

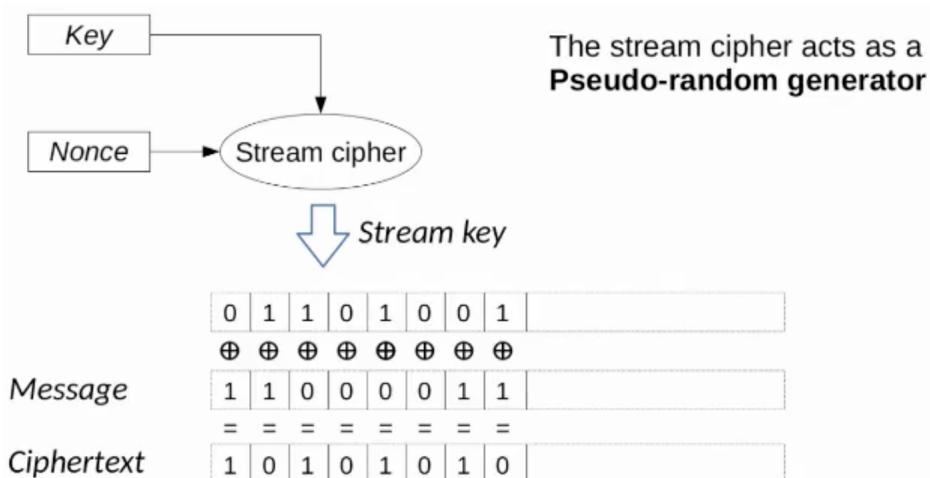
Se il costo per rompere lo schema è minore della brute force (causa falla nello schema), allora si deve deprecare completamente l'algoritmo, in quanto è considerato sicuro fino a che il costo per romperlo è considerabile come esponenziale.

Stream Cipher

Concettualmente simile al One Time Pad, ma al posto di avere una chiave completamente random abbiamo quella che viene chiamata *string key* o chiave a flusso che è una chiave pseudo random.

Dato pseudo random: È un dato che viene generato da una così detta funzione, che è un generatore pseudo random, questo prende in input una piccola quantità di dati random e genera una sequenza di bit di grandi dimensioni che è indistinguibile da dati random.

Dato random: Generato attraverso processi fisici che vengono utilizzati per generare dati random e sono buoni generatori perché totalmente unpredictable.



La chiave che va conosciuta non è la chiave lunga quanto il messaggio (*stream key*), ma la piccola chiave che diamo come input allo stream cipher. Questo stream cipher può espandere questi dati random in una sequenza di grandi dimensioni.

Attenzione: questa funzione è comunque una funzione deterministica. Se noi eseguiamo questa funzione con gli stessi input otteniamo sempre la stessa stream key, inoltre eredita anche i problemi di malleabilità di one time pad.

Multi message security: posso voler utilizzare la stessa chiave per cifrare più dati. Il problema degli schemi di cifratura è che quando lo usiamo dobbiamo dare una garanzia: anche se cifro lo stesso messaggio l'output deve essere sempre differente, l'attaccante non deve poter discernere lo stesso tipo di dati.

La soluzione e' il parametro **nonce**, garantisce che la stringa generata è sempre diversa dalla stream key precedente. Questo parametro, a differenza della chiave, non necessita di essere confidenziale.

Se si utilizza lo stesso *nonce* per ogni messaggio, l'attaccante ottiene due messaggi cifrati che può mettere in relazione e ottenere, tramite proprietà matematiche, moltissime informazioni sui dati in chiaro: distribuzioni caratteristiche, ... una catastrofe.

- Stream cipher nativi
- Stream cipher derivati da block cipher

Praticamente tutti gli stream cipher nativi sono deprecati ad eccezione di pochissimi tra cui ChaCha20.

Block Cipher

Un block cipher e' un oggetto che ci dà garanzie a livello matematico, e' una famiglia dipendente da una chiave di permutazioni pseudorandom.

AES e' un famiglia di cifrari a blocchi, identifica tre particolari istanze:

- AES-128
- AES-192
- AES-256

Ogni numero identifica la lunghezza in bit della chiave. Per utilizzare uno schema di cifratura bisogna aggiungere un altro acronimo alla sequenza, la *modalità di cifratura* ad esempio AES-128-CBC.

Il block cipher alla fine è solamente un cifrario di sostituzione, non lavora con l'alfabeto ma con dati binari, nello specifico è un cifrario di sostituzione ideale poli alfabetico fatto di zeri e uni.

The table is the ideal block cipher

- a raw is transferred as the key?
 - $n * 2^n$ → too large (grows exponentially and thus it is not efficient)
- if the table is public and the index of the raw is the key
 - $n * 2^n * 2^n!$ → it is impractical to store the table or to re-compute the raw

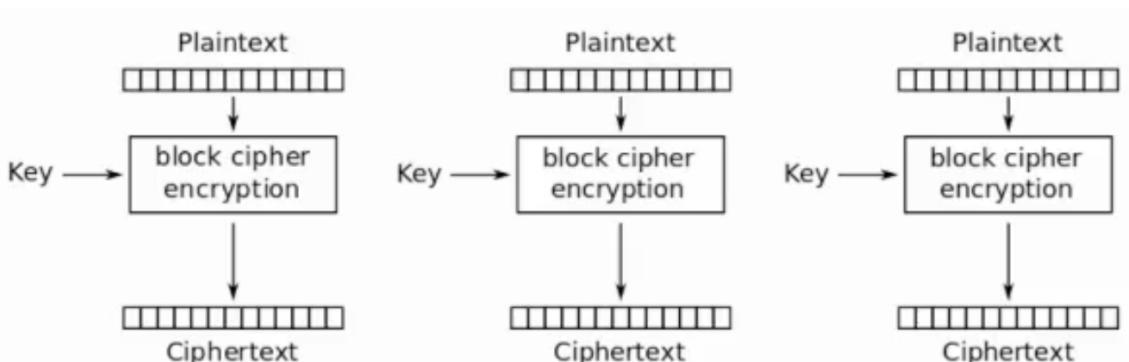
Idealmente il problema di dover memorizzare tutte le permutazioni si propone nuovamente infatti n compare all'esponente non permettendo un costo divergente, l'attore legittimo ha già un costo esponenziale.

Nella realtà non ci serve memorizzare tutte le permutazioni ma ci basta una chiave.

00 01 10 11	00 01 10 11	00 01 10 11	00 01 10 11
0: 00 01 10 11	6: 01 00 10 11	12: 10 00 01 11	18: 11 00 01 10
1: 00 01 11 10	7: 01 00 11 10	13: 10 00 11 01	19: 11 00 10 01
2: 00 10 01 11	8: 01 10 00 11	14: 10 01 00 11	20: 11 01 00 10
3: 00 10 11 01	9: 01 10 11 00	15: 10 01 11 00	21: 11 01 10 00
4: 00 11 01 10	10: 01 11 00 10	16: 10 11 00 01	22: 11 10 00 01
5: 00 11 10 01	11: 01 11 10 00	17: 10 11 01 00	23: 11 10 01 00

Questa chiave va ad agire come index , ci indica quale sostituzione si deve utilizzare.

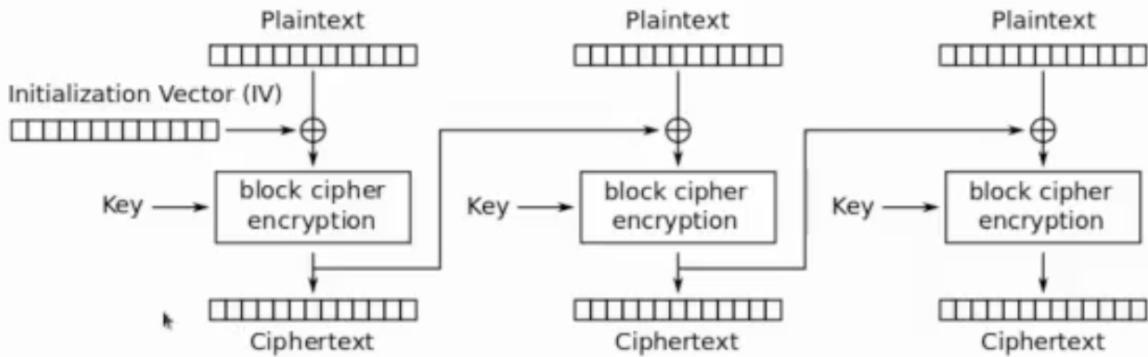
I blocchi sono grandi 128 bit



Se si utilizza la pura sostituzione(ECB), quindi suddividendo la dimensione del file in blocchi da 128 bit e criptando ogni blocco si può incorrere in falle di sicurezza in quanto pattern di criptazione potrebbero essere visibili.

Quindi solitamente non si utilizza direttamente il block cipher ma si aggiunge la **mode encryption** ovvero una modalità aggiuntiva con cui si costruisce lo schema di cifratura robusto.

Una modalità di criptazione molto utilizzata è Cipher Block Chaining(CBC)



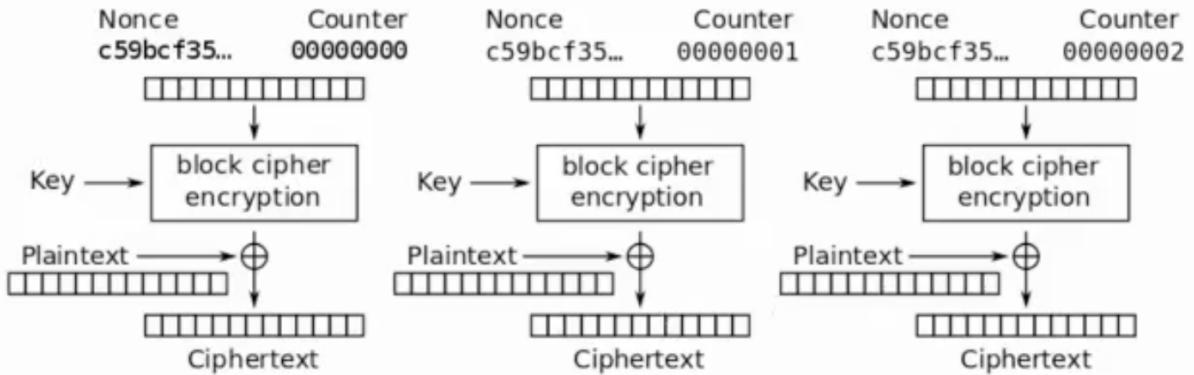
Ogni blocco 'dipende' da quello precedente, infatti viene eseguito uno xor tra il plain text e parte del blocco cifrato in precedenza. L'**initialization vector** è un oggetto simile al *nonce* (ma grande quanto il blocco), ci serve se vogliamo garantire la sicurezza di tanti messaggi, se si cifrassero testi uguali senza l'*initialization vector* si otterebbe lo stesso output.

Nonce e *Initialization vector* sono termini intercambiabili, ad esempio si potrebbe trovare uno stream cipher che vuole in input un initialization vector. In generale i requisiti di sicurezza sono:

- stream cipher, priorità e l'unicità
- CBC, priorità e la randomizzazione

Nonostante in CBC si cifra un blocco alla volta, possiamo utilizzare un cifrario a blocchi per creare uno stream cipher.

La modalità più nota per utilizzare un block cipher come uno stream cipher è la *Counter Mode* (CTR)



Il testo in chiaro non è mai fornito come input al *block cipher*, in particolare il block cipher viene utilizzato come un generatore di dati pseudorandom. Prende in input un oggetto(grande quanto il block size) composto da due parti

- *nonce*
- *counter*, ogni volta che si invoca il block cipher si incrementa il contatore

Le dimensioni del *nonce* e del *counter* vengono scelte in base al contesto, quando si devono cifrare dati di grandi dimensioni il counter è più grande, al contrario nel caso in cui bisogna cifrare tanti messaggi di piccole dimensioni.

L'output del block cipher diventa la *stream key*

In CTR posso permettermi di troncare la stream key a seconda di quanti bit devo cifrare, nella modalità CBC devo cifrare blocchi rigorosamente di 128 bit quindi l'input deve essere un multiplo.

Padding

Se devo cifrare dati di cui la dimensione non e' un multiplo di 128 bit si utilizza il **padding**, dati fintizi riconoscibili per fare in modo che l'input sia multiplo della *block size*. Lo standard più utilizzato e' il PKCS7

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
h	e			o	w	o	r		d	6	6	6	6	6	6

- Add N times the value N of the missing number of bytes
- If the data is already padded, add block-size times the value of the block-size

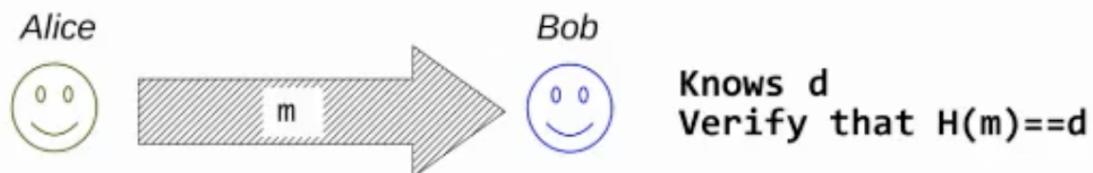
Se ad esempio mancano 6 byte per completare il blocco, questo lo riempio di 6.

Il padding non rivela informazioni perché anch'esso viene cifrato.

La cifratura non ha l'obiettivo di nascondere la dimensione dei dati, nel caso sia necessario nascondere la dimensione del messaggio (ad esempio in campo militare)

'yes' o 'no') bisogna manipolare il dato prima di inviarlo, quindi utilizzare il padding a prescindere.

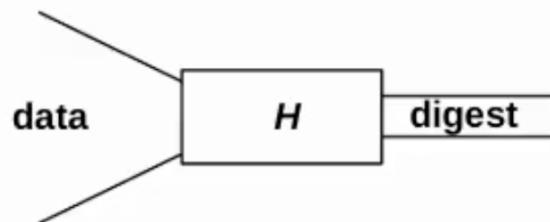
Hash Functions



Le funzioni *hash* permettono di controllare l'integrità dei dati, quindi assicurarsi che questi non siano stati manipolati. L'integrità spesso viene anche riferita al concetto di *autenticità*, cioè al fatto che sia stato un determinato utente a mandare il messaggio.

$$H : \{0,1\}^* \rightarrow \{0,1\}^n$$

hash(data) → digest

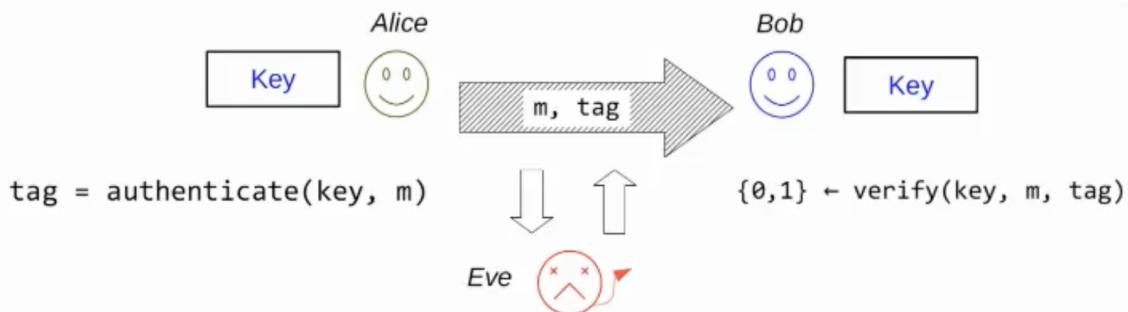


In linea teorica una funzione hash non può essere sicura in quanto restituisce un hash di lunghezza prestabilita prendendo in input dati di qualsiasi lunghezza. Quindi inevitabilmente alcuni input restituiscono lo stesso output => *collisioni*. Una funzione hash e' sicura fin quando (nonostante l'algoritmo sia pubblico) non si trovano modi per ottenere collisioni.

L'attacco compleanno è il più efficiente nei confronti delle funzioni hash, si basa puramente su calcoli statistici. Se abbiamo n elementi, il numero di tentativi per avere il 50% di probabilità di trovare una collisione è la radice quadrata di n. Infatti nel calcolo del *livello di sicurezza* si divide sempre a metà il numero di bit del digest, es. se ho una funzione hash che genera un digest di 256 bit il livello di sicurezza è 128 bit.

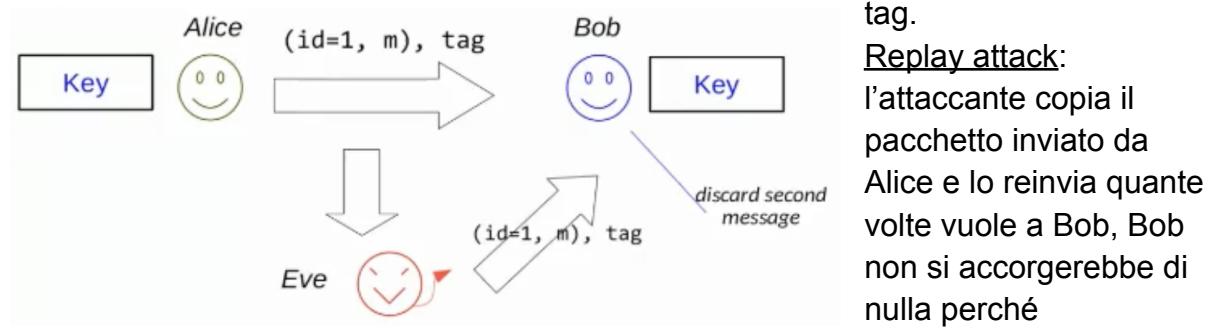
sha1 deprecato nel 2005 => nasce *sha2* mettendo qualche pezza all'algoritmo di *sha1*, nel mentre si lancia un contest per la creazione di *sha3* creando un algoritmo totalmente nuovo pensando che da lì a breve si trovassero collisioni in *sha2*. Questo non e' accaduto, *sha2* e *sha3* sono le funzioni hash attualmente sicure.

Message Authentication Codes



Per garantire l'autenticità si usa la funzione MAC. Questa funzione accetta due input, la chiave e il messaggio, in output produce un tag: in modo simile alle funzioni hash, è un dato di dimensioni costanti a prescindere dalla dimensione del messaggio. In pratica MAC e' come una funzione hash segreta, il cui algoritmo non è noto all'attaccante. Questa segretezza non permette di calcolare il tag.

Il livello di sicurezza è determinato innanzitutto dalla dimensione della chiave. Non è consigliabile troncare il tag.



pacchetto e' autenticato. Se ad esempio il messaggio contenesse una transazione di soldi verso un conto questa verrebbe eseguita n volte causando non pochi problemi.

Per ovviare a questo problema si può inserire un *id* che viene incrementato ad ogni invio, così da permettere al destinatario di tenere traccia dei messaggi già ricevuti.

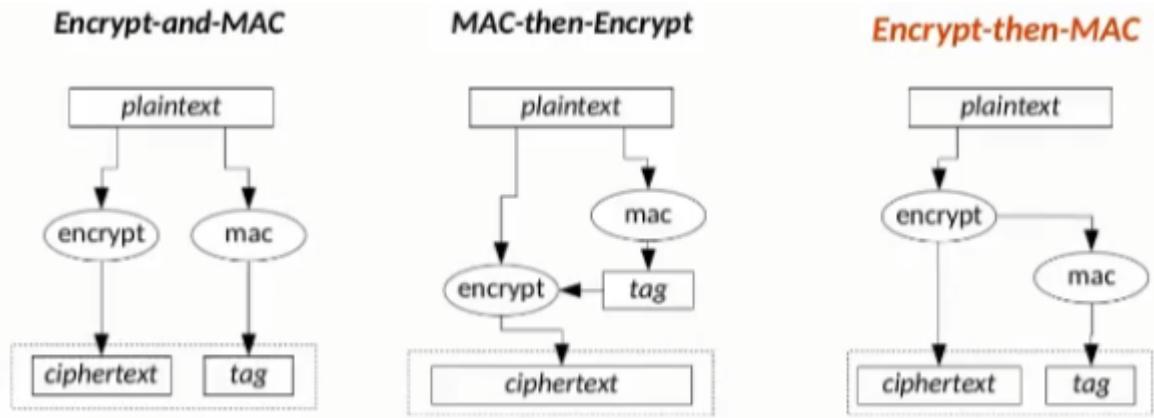
```
from secrets import compare_digest

def authenticate(key, message):
    mac = HMAC(key, digestmod=sha256)
    mac.update(message)
    return mac.digest()

def verify(key, message, tag):
    mac = HMAC(key, digestmod=sha256)
    mac.update(message)
    #return mac.digest() == tag
    return compare_digest(mac.digest(), tag)
```

Funzioni *HMAC* sono basate su funzioni hash. Il confronto fra i due tag deve essere a tempo costante, non si può usare ad esempio l'operazione `==` ma bisogna utilizzare funzioni apposite come `compare_digest`, questo per evitare *timing attack* basati sul tempo di esecuzione.

Per ottenere un **Authenticated encryption scheme** bisogna combinare la criptazione simmetrica e il MAC, si può fare in vari modi:



In generale il metodo più sicuro è **encrypt-then-MAC**, le funzioni MAC sono deterministiche (non hanno un *nonce*) quindi è necessario che si basino su testi randomizzati. Il problema principale del terzo paradigma è la performance.

Tre regole importanti da seguire per **encrypt-then-MAC**:

1. The MAC must authenticate the ciphertext and the nonce, vogliamo essere sicuri che l'attaccante non abbia manipolato l'initialization vector e il testo cifrato.
2. The symmetric encryption scheme and the MAC must use different cryptographic keys.
3. In the decryption phase, the MAC must be verified before starting to decrypt the ciphertext.

AEAD schemes: schemi di cifratura autenticato, vengono garantite entrambe le caratteristiche: cifratura + MAC. es.

AES-GCM, AES-GCM-IV => CTR + GMAC oppure
Chacha20Poly1305 => Stream + Poly1305

Questi schemi sono comodi per evitare errori a livelli di progetto della generic composition e come performance sono ottimizzati rispetto ad un implementazione artigianale di encrypt then mac.

AEAD – Authenticated Encryption with Associated Data

Utilizzato praticamente in tutti i sistemi moderni. Questo framework è pensato per offrire tutte le funzionalità con diverse sfaccettature.

`keygen([size]) → key`

p → plaintext
a → associated data
c → ciphertext
n → nonce

`encrypt(key, n, a, p) → c`

Note: here we use nonce as an example, but a randomized IV might be needed depending on the underlying scheme.

`decrypt(key, n, a, p) → m`

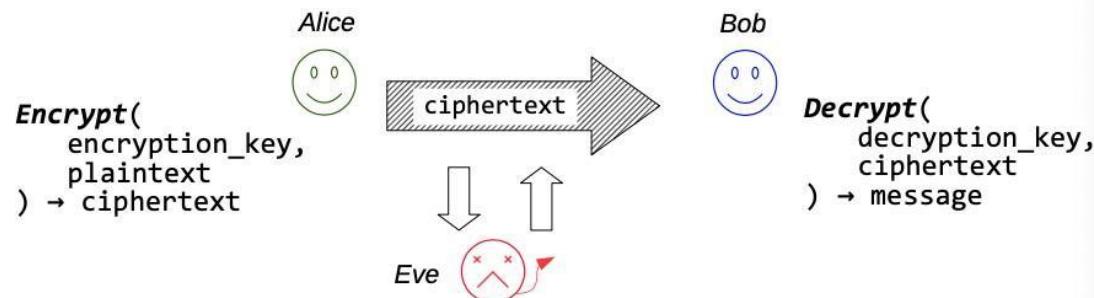
Ad esempio nel parametro `a` posso indicare i dati associati ovvero dati cui voglio garantire l'autenticità ma non la confidenzialità.

Quando creiamo delle applicazioni è molto comune avere dei dati che strutturalmente non possiamo cifrare, perché sono fondamentali per il funzionamento del protocollo: i metadati, se ho un header ne voglio garantire l'autenticità, ma non possiamo cifrarli.

Dati random sono presi generalmente da eventi fisici, ad esempio correnti residue presenti nella CPU, se non sono disponibili a livello integrato le istruzioni per accedere a queste informazioni in molti server vengono installati dei dispositivi hardware che hanno questo ruolo.

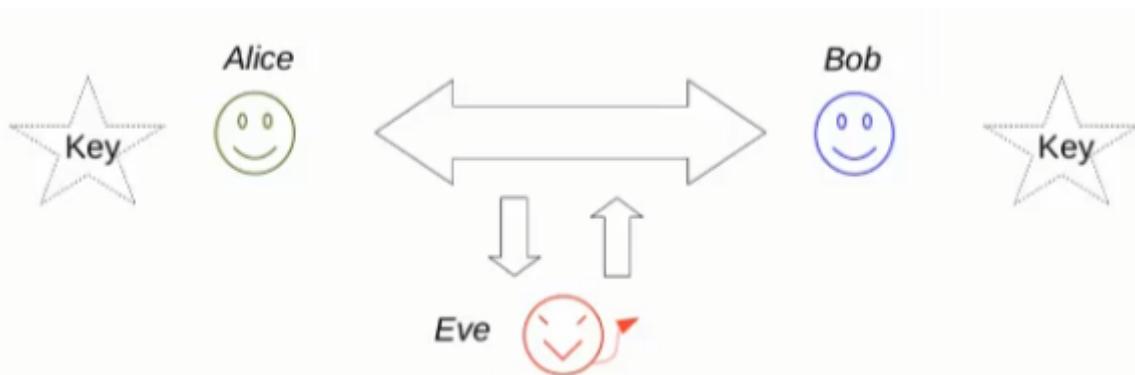
Non tutti i random sono uguali, ad esempio su Python nel modulo random e' specificato nella documentazione che non può essere usata a livello di crittografia in quanto simula un ambiente random.

Crittografia asimmetrica



In un contesto asimmetrico le chiavi di cifratura e decifratura sono diverse. In precedenza abbiamo assunto che Alice e Bob si fossero scambiati le chiavi in maniera sicura, la crittografia asimmetrica inizialmente e' nata per risolvere questo problema.

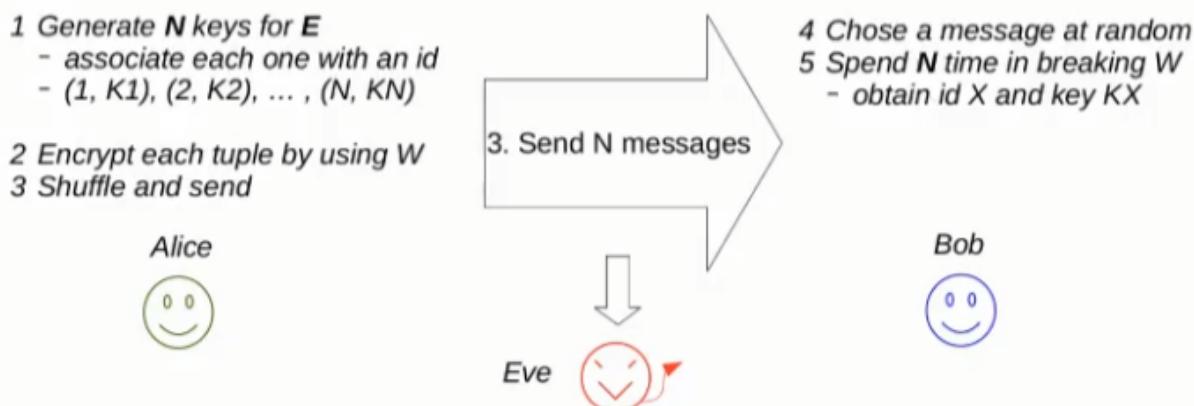
Key exchange protocol



L'obiettivo e' far ottenere ad Alice e Bob la chiave(uguale per entrambi) senza che Eve possa leggerla.

Merkle Puzzle (1974)

Protocollo di scambio di chiavi basato su crittografia simmetrica. Consideriamo che abbiamo chiavi uguali e schemi noti.



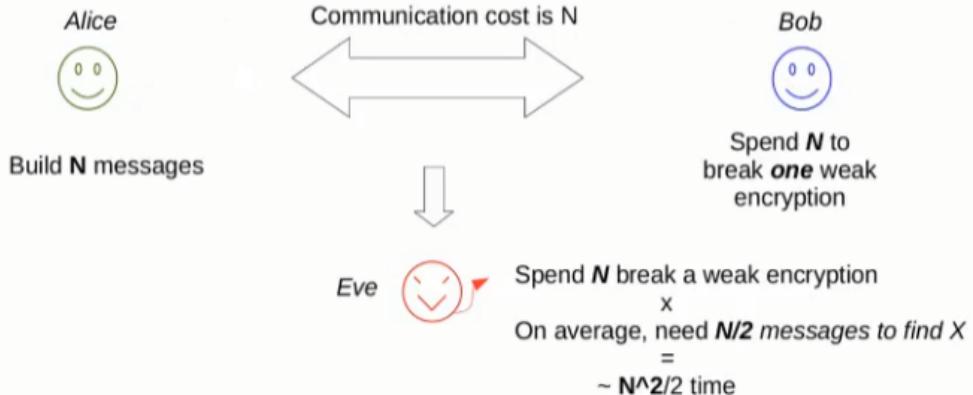
Nello specifico abbiamo due schemi:

- Schema E , che non può essere rotto
- Schema W , che può essere rotto.

Alice genera n chiavi che possono essere utilizzate per E . Cifra ciascun elemento (ID, Chiave) usando lo schema debole W . Fa uno shuffle e le manda a Bob.

Bob riceve gli N messaggi e ne sceglie 1 a caso e rompe la cifratura ottenendo il dato in chiaro.

Bob risponde che ha scelto l'id X , Alice si era mantenuta tutto il db delle key generate è in grado così di risalire alla chiave k tramite l'id che le e' arrivato da Bob.



Supponendo che per rompere lo schema si spenda N , Eve in media ci metterà $N/2$ per trovare la chiave giusta così avendo un costo di tempo $N/2 * N$ quindi divergente rispetto al costo(N) di Bob.

Questi costi però divergono troppo poco per utilizzare il protocollo nel mondo reale, non sono comparabili ai costi quasi esponenziali della *computational security*. L'idea è quella di costruire *trap door* (problemi matematici) che sono efficienti da eseguire in una direzione ma inefficienti da eseguire all'inverso.

Example:

$$\mathbb{Z}_{11} \rightarrow \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$$

$$3^7 \bmod 11 = 9$$

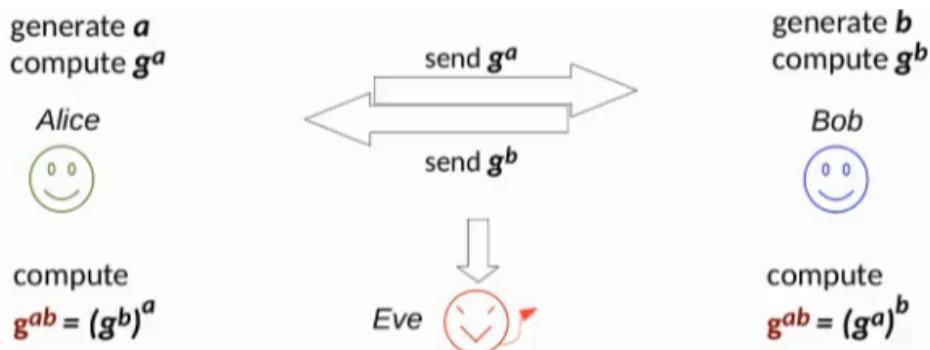
given the base 3 (the generator) and the result 9 it is "hard" to compute the exponent 7

La costruzione matematica più diffusa è *il problema del logaritmo discreto*, l'operazione diretta è efficiente invece non c'è soluzione efficiente per il calcolo dell'esponente (dato 9, il modulo 11 e la base 3) è definita come un'operazione "hard" quindi con un costo quasi esponenziale.

Soltamente si astrae, quindi in generale la base(il 3 di prima) è detto generatore g , il numero degli elementi del gruppo ciclico è p (11 nel caso di prima).

Quando si fa riferimento ad g^a si sottintende diviso il modulo(p) in quanto ha senso solo con i numeri interi e non con le curve ellittiche.

Diffie-Hellman



Il protocollo funziona che ogni attore(Alice e Bob) si calcola uno scalare random a & b .

1. Gli attori calcolano g^a e g^b .
2. Alice manda a Bob g^a e Bob g^b .
3. Alice, visto che conosce a , può calcolarsi g^{ab} . Lo stesso fa Bob. Eve, invece, non riesce a calcolare g^{ab} .

Numeri interi => DH

Curve ellittiche => ECDH

Vengono utilizzate le curve ellittiche perché prendono input più piccoli per ottenere un livello di sicurezza alto rispetto ai numeri interi che a parità di livello di sicurezza bisogna dare in input numeri molto grandi. Infatti i protocolli con le varianti intere sono praticamente deprecati.

La curva P224 ha un livello di sicurezza a 112 bit, la P256 a 128 bit, ... come le funzioni hash.

Digital signatures(firme digitali)

Sono quelle che le funzioni che nell'ambito asimmetrico mi permettono di garantire l'autenticità dei dati (come le funzioni MAC per l'ambito simmetrico). La funzione di autenticazione si chiama "firma". Firma e verifica non si basano sulla stessa chiave ma su chiavi diverse (secret e public).

`sign(sk, message) → signature`

$pk \Rightarrow$ public key

$sk \Rightarrow$ secret key

`verify(pk, message, signature) → {true, false}`

In uno schema MAC per costruzione la sk è nota PER FORZA da due persone, mentre qua la chiave la conosce solamente il mittente.

Quindi se quella persona ha prodotto una firma e questo può essere fatto solamente da chi ha quella determinata chiave, non si può ripudiare in alcun modo questa cosa.

Distribuire la chiave pubblica non intacca la segretezza dei dati

Gli algoritmi visti in precedenza per lo scambio di chiavi sono vulnerabili all'attacco MITM(Man in the middle).



Entrano in nostro soccorso le firme digitali(che garantiscono non repudabilita'), Alice oltre ad inviare g^a invia anche la firma applicata a g^a (Bob deve conoscere la public key di Alice). *key pair*, termine usato per indicare la coppia di chiavi (segreta-pubblica)

Dell'associazione tra identità e numeri per i calcoli crittografici se ne occupano le architetture che distribuiscono le chiavi.

Se vogliamo utilizzare una firma digitale che si basa sullo stesso problema matematico del protocollo Diffie Hellman(logaritmo discreto/curve ellittiche) lo standard e' DSA(Digital Signature Algorithm) schema alternativo al RSA. Nello specifico si utilizza la variante con le curve ellittiche *ECDSA*.

Le firme digitali possono essere *deterministiche* o *randomizzate* a seconda del protocollo di firma digitale utilizzato, schemi come DSA o ECDSA per essere eseguiti correttamente richiedono ad ogni operazione di firma di generare un numero random(k), esistono versioni che generano k in maniera automatica (RFC6979).

EdDSA sono schermi di firma non basati sugli standard del NIST, pensato per essere deterministico quindi non richiede dati random(usato da ssh,tor,...), firme dovendo garantire solo autenticita' possono permettersi di essere deterministiche .

RSA

Textbook RSA

RSA e' in realtà un problema matematico al livello del logaritmo discreto, solo che si basa su moduli che e' il prodotto di numeri primi.

- $n = p * q$
 - n is the public modulus
 - p and q are the secret primes
 - Compute k, d such that
 - $c = m^k \text{ mod } n$
 - $m = c^d \text{ mod } n$
- Avendo p e q e il prodotto n , possiamo calcolarci due valori k e d che se usati come esponenti riescono a fornirci due operazioni: una per passare da m a c e viceversa. A seconda dell'esponente che non si conosce non si riesce a calcolare c o m

Il problema è leggermente diverso concettualmente del logaritmo discreto che ci da un'operazione unidirezionale NON invertibile. Questo problema è invertibile, ma che riesco a rendere unidirezionale e non invertibile sulla base delle informazioni segrete che rendo note.

Protocollo RSA

In realtà *RSA* è composto da algoritmi di criptazione e firma basati sullo standard *PKCS* in cui non c'e soltanto il calcolo matematico ma anche la trasformazione dei dati tramite uno schema di padding.

RSA essendo uno schema invertibile può essere utilizzato sia per generare firme (utilizzando la chiave privata) sia cifrare in maniera asimmetrica.

Encryption

- PKCS-OAEP
- PKCS1-v1.5

Signing

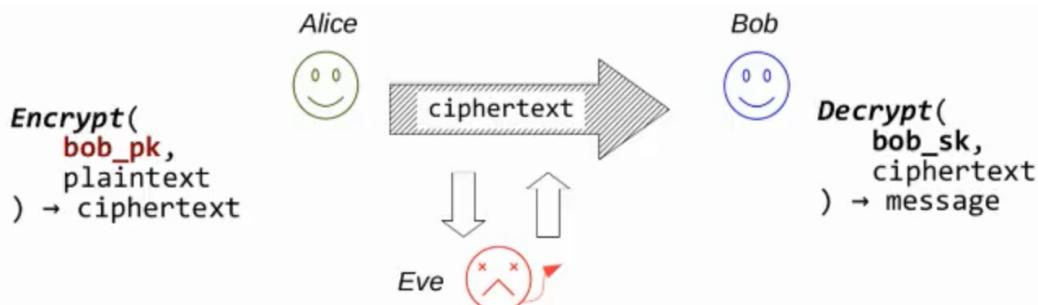
- PKCS1-PSS: modello più nuovo che implementa la funzione H, che sono all'interno funzioni di hashing e fanno operazioni di varia natura; sono chiamate nel mondo asimmetrico schemi di padding(da non confondere con quelli nel mondo simmetrico), il concetto di padding è di adattamento; in questo caso non devo far diventare dati multipli di una certa dimensione, ma trasformare il dato in un numero che possa essere utilizzato in questa operazione di elevamento a potenza.
- PKCS1-v1.5: questo è il protocollo storico, che troviamo praticamente sempre a volte ancora adesso.

RSA esiste solo su numeri interi, non sono presenti varianti(come le curve ellittiche).

In contesti in cui si hanno dispositivi che devono generare tante firme e' generalmente meglio *ECDSA*, invece se si hanno poche firme ma molte verifiche *RSA* potrebbe essere la soluzione giusta, rimane il fatto che più il livello di sicurezza aumenta più le curve ellittiche sono dominanti rispetto ai numeri interi.

Cifratura asimmetrica

Si può effettuare uno scambio di chiavi senza Diffie-Hellman? Si, con la criptratura asimmetrica.



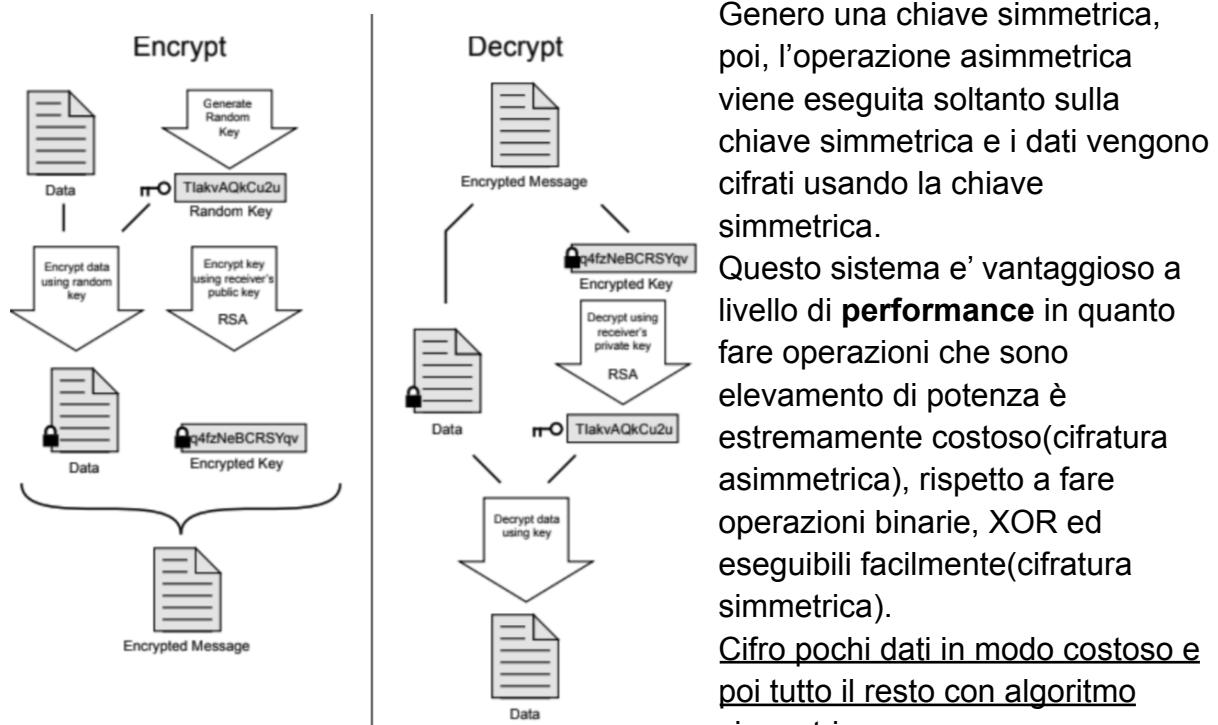
Alice utilizza una chiave pubblica di Bob per criptrare dati destinati a lui. Nella criptratura chi cifra ha la chiave pubblica => solo chi ha la chiave privata può decifrare.

L'idea di base:

- Criptazione: Alice usa la chiave pubblica
- Firma: Bob usa la chiave privata

Esistono schemi di firme digitali e schemi di criptratura asimmetrica che sono completamente indipendenti. Se vogliamo fare una firma con RSA devo usare PKCS1-PSS e se devo criptrare uso PKCS1- OAEP.

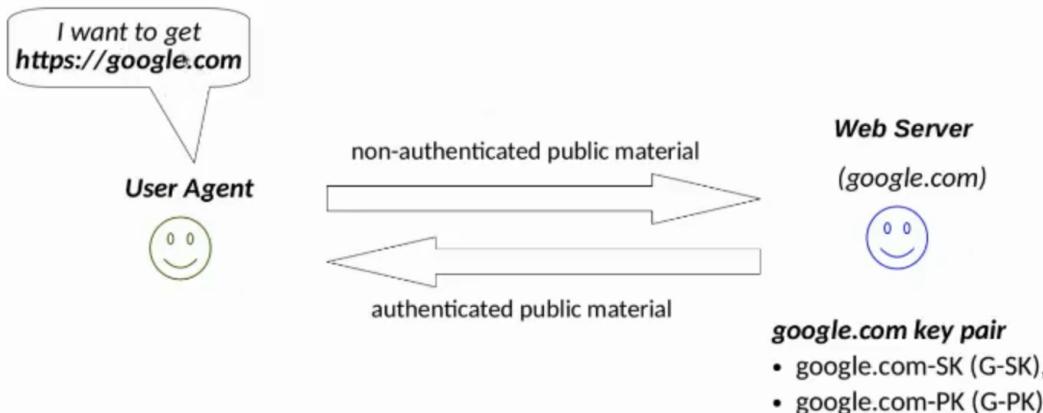
La criptratura asimmetrica del mondo reale è detta ibrida perché la usiamo insieme ad una criptratura simmetrica.



Inoltre la criptratura asimmetrica non è sicura in alcuni scenari...

Public Key Infrastructure

Ha come obiettivo risolvere l'ultimo problema che ci manca: la distribuzione della chiave pubblica.



Noi vogliamo autenticare delle chiavi pubbliche che riceviamo dal server rispetto all'identità google.com, con le email saranno gli indirizzi email, ecc... l'identità è l'informazione di alto livello che l'utente inserisce per identificare con chi vuole comunicare.

Abbiamo diversi approcci:

TOFU(Trust on first use)

Facciamo uno scambio di chiave non autenticato. La prima volta che comunico con una persona/entità non ho la sua chiave autenticata e assumo per quella prima comunicazione io non sia sotto attacco, "ci fidiamo".

Ad esempio SSH, se provo a connettermi il server ssh mi chiede se sono sicuro di volermi connettere a quel server, quindi assumo che nessuno stia facendo un attacco MITM, tutte le prossime volte si verifica la stessa chiave pubblica ottenuta la prima volta. Nel caso in cui stia subendo un attacco MITM comunque rimarrebbe la segretezza dei dati in quanto viola solo l'autenticità di questi.

Out of bounds communications

Quando riusciamo ad usare 1 o più canali di comunicazione oltre a quello su cui vogliamo comunicare. Due canali di comunicazione

- sicuro(solitamente più lento), dove si scambia la chiave pubblica
- insicuro, scambio i dati dopo lo scambio delle chiavi

Per canale di comunicazione secondario posso intendere anche canali di altra natura: una telefonata anche all'altra persona per dire. Così alla fine sono certo che solo chi di dovere possa decifrare i miei dati.

Ad esempio Whatsapp, il software client tramite la fotocamera verifica l'identità della chiave scannerizzando il QR code fornito.

Delegated approaches

Si comunica con un solo canale però qualcuno al posto nostro verifica l'autenticità delle chiavi tramite firme. Questo e' l'approccio più scalabile pero' e' complesso perché aggiungo un problema di fiducia in quanto mi fido di persone che mi firmano le chiavi pubbliche e devo amministrare tutta l'infrastruttura per gestire lo schema.

Approcci principali:

- Metodo centralizzato: ci fidiamo un ente terzo PKI.
- OpenPGP Web of Trust(standard di email cifrate e altri servizi)

Il difetto di questi approcci delegati è sempre il: di chi mi fido? Come gestisco la fiducia?

Dipende tutto sempre dal contesto. Il contesto di WhatsApp esiste da decenni per il software OpenPGP per le mail cifrate. In questo tipo di applicazione si può realizzare quella che viene chiamata la **web of trust**, rete di delegazione di fiducia (tra persone alla pari che si sfruttano per verificare informazioni) e mi fido di questo qualcuno per ottenere le chiavi autentiche.

PKI è un approccio delegato di tipo centralizzato: noi abbiamo una centralizzazione forte della delegazione della fiducia. Creiamo questo soggetto: terza parte fidata. È una persona di fiducia che è una società o entità predisposta per questo ruolo. Questo rimane un punto debole, parti fidate non si sono rivelate sempre fidate.

Automatic verification approaches

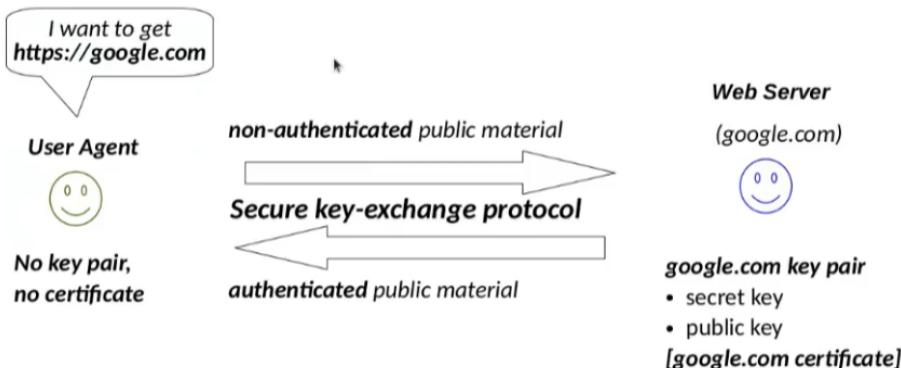
Voglio verificare un identità' che ha un ruolo nella comunicazione che voglio realizzare, ad esempio se in ambito web voglio parlare con google.com questo hostname e' identificabile tramite servizio DNS. Quindi realizzare degli approcci automatici che facilitano la verifica delle key pubbliche sfruttando il fatto che sono identità' importanti nella comunicazione.

Questo tipo di situazione ha fatto nascere ACME che permette il rilascio automatico di certificati web per verificare automaticamente la corrispondenza fra un dominio internet e la sua chiave pubblica. Si mira a controllare il fatto che: chi dice di essere un determinato dominio possa essere verificato in maniera certa.

Le key pair hanno metadati (es.l'identità), l'insieme di tutti i metadati prende il nome di *certificati*.

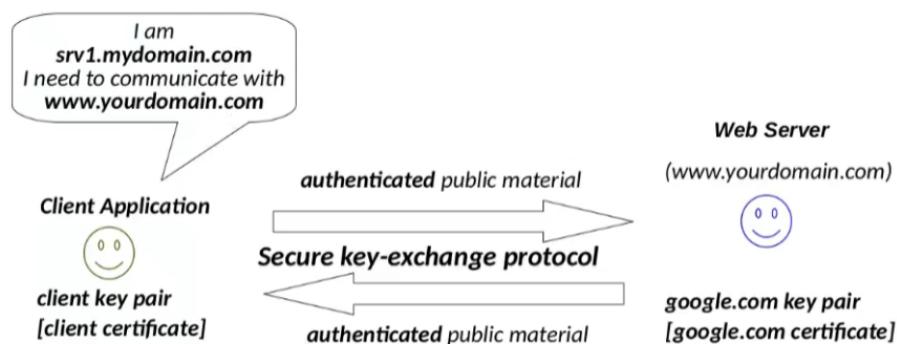
Possibili scenari

User -> Server



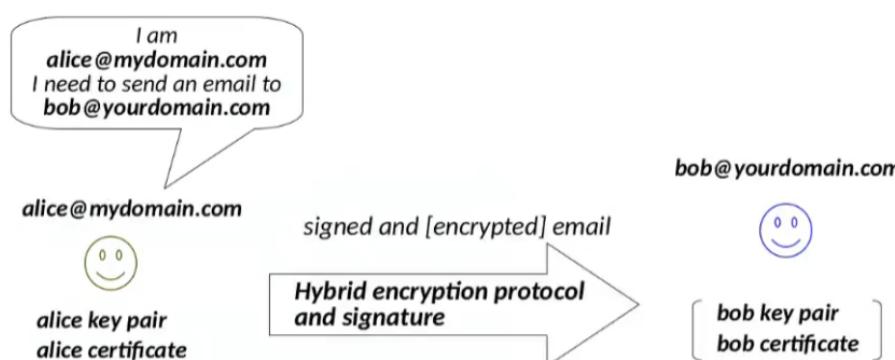
Il lucchetto verde sui siti è sinonimo del fatto che le comunicazioni siano sicure, riesco a verificare che chi mi ha risposto è effettivamente il sito di Google e non qualcun altro. È una comunicazione unidirezionale, il client non viene autenticato.

Due servizi



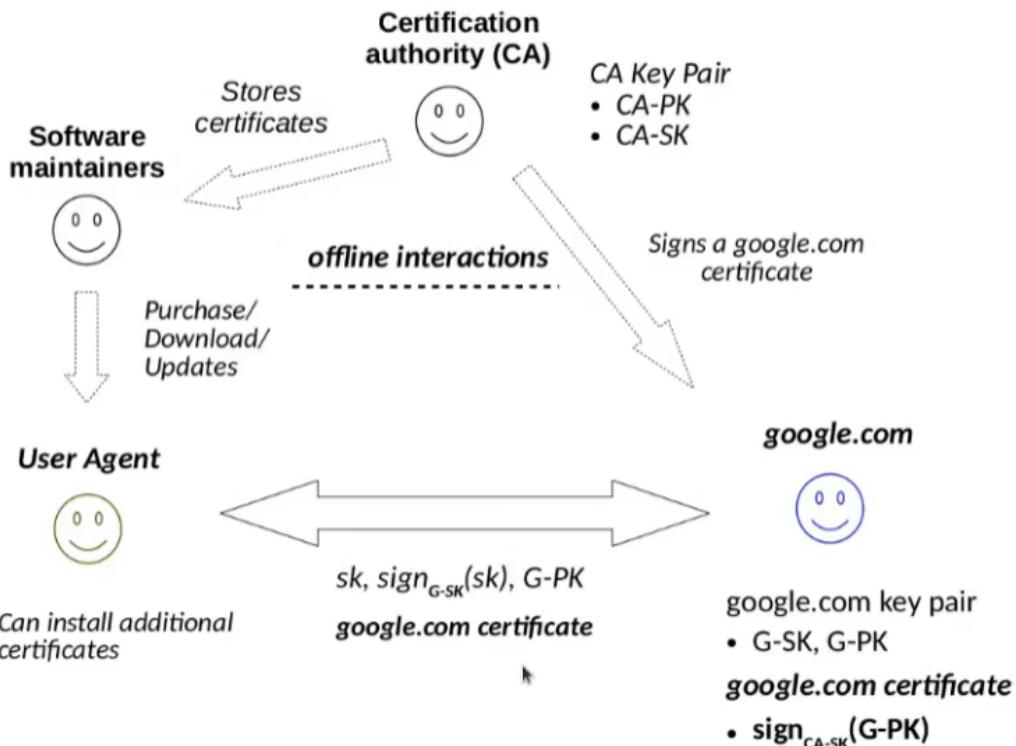
Voglio autenticare entrambe le direzioni: quando voglio creare comunicazioni non fra utente e server, ma fra due servizi. Immaginiamoci che il client sia eseguito su un altro sito: due servizi differenti che usano API per fare servizi collaborativi di varia natura.

Email



In questo caso l'obiettivo dell'autenticazione non è ottenere scambio di chiavi sicure, ma usare correttamente uno schema di cifratura e firma (autenticare messaggi),

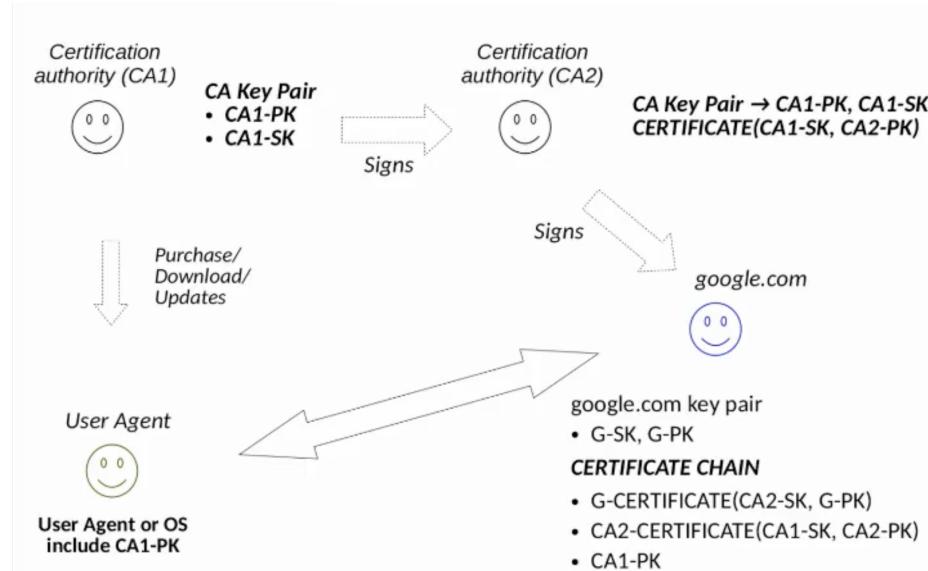
autentico uno schema di cifratura e firma)



Ci sono delle aziende CA (Certification Authority) che certificano che le persone giuste abbiamo i certificati giusti. Questa è un'azione che viene fatta a livello legale. Controllo proprio l'identità legale di chi controlla questi servizi.

Noi ci fidiamo di questa catena di certificati (certifications chain) perche' nei nostri user-agent abbiamo preinstallato le chiavi pubbliche delle *certifications authorities*. A sua volta, ad esempio Mozilla, installa quelle chiavi perché ci sono dei consorzi che stabiliscono quali certifications authorities sono sicure.

Certificate chain



Nel web troveremo un *Certificate Authority Root*, una intermedia e un web server. La differenza è: come prima quando ci connettivamo a Google avevamo i materiali DH ci manda in aggiunto il fatto che CA2 è stata firmata e usata da CA1.

Nel client ho poche chiavi e non si salvano tutte le CA intermedie per motivi di gestione e scalabilità.

Le CA che sono in cima alla gerarchia sono chiamate Root CA. Non ce n'è solo una ma ce ne sono varie. Che cosa definisce quali sono le root CA? Nessuno, ci basiamo su della policy che possono anche variare leggermente in base a chi le definisce.

Tipi di Certification Authorities

- **public**: rilasciate da aziende pubbliche in maniera chiara dietro compenso.
- **internal**: CA autorizzata da una root pubblica ma ad uso privato.
- **private**: utilizzate all'interno di un'azienda, es. Google, rilascia i certificati dei propri server attraverso le proprie certifications authorities. Questo tipo di approccio e' possibile solo in aziende molto grandi.

x509 Certificate

Materiale binario in base64 delimitato da *begin* e *end*.

Ogni certificato contiene varie informazioni:

- protocolli di cifratura utilizzati
- periodo di validità, data di inizio e di scadenza
- il tipo di certificato, un certificato che viene utilizzato per un server web non può essere utilizzato per la posta elettronica
- issuer, chi ha rilasciato il certificato
- serial number, numero identificativo

Il flow per ottenere un certificato e' questo:

1. Server:
 - a. Create a Valid Key Pair
 - b. Generate CSR(Certificate Sign Request): con questo oggetto il server fa la richiesta ad una CA per il rilascio del certificato, il server mette solo la PK in una struttura dati in cui chiede alla CA di firmarla.
2. Issue the CSR to the CA, la CA dopo aver verificato le informazioni nel CSR valuta che sono valide allora rilascia il certificato.
3. Securely store the secret key on the Web Server

Tipi di certificati

Domain validated (DV)

Identifica un certificato in cui la CA nella fase di validazione ha solo controllato che il dominio inserito nel CSR e' effettivamente sotto il controllo di chi ha chiesto il certificato.

Extended validation (EV) & Organization validated (OV)

La certification authority non controlla solo informazioni tecniche ma anche legali.
Viene utilizzato il certificato EV rispetto al OV.

Revoca di un certificato

Rendere non più valido un certificato rilasciato in precedenza, questo può essere utile in caso di chiave rubata oppure semplicemente ho dismesso il servizio a cui è legato.

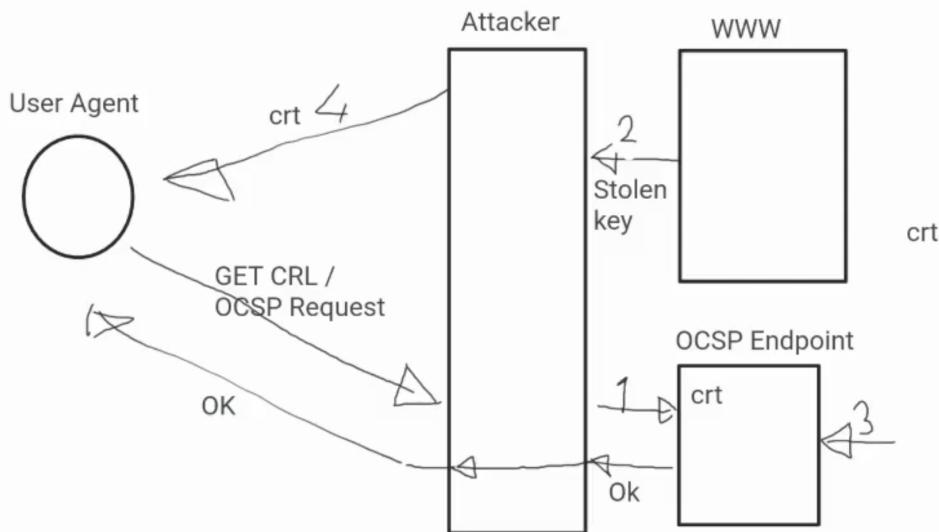
Certificate Revocation Lists (CRL): Distribuisco una lista di certificati non più validi.

Online Certificate Status Protocol (OCSP): Protocollo per effettuare operazione di verifica su un file CRL, al posto di dire "dammi il file CRL" ho un API a cui devo chiedere se il certificato e' stato revocato o meno.

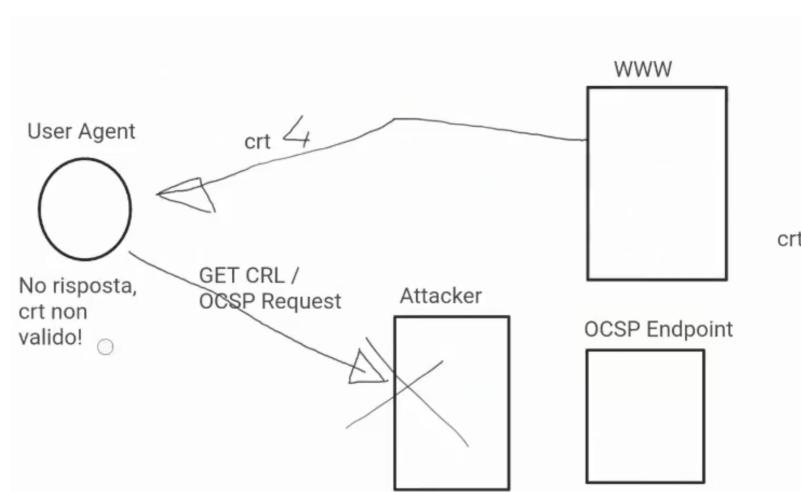
Fino a qui abbiamo visto che PKI funziona correttamente anche in casi di revoca, però abbiamo due aspetti fondamentali:

- Il sistema di revoca è facilmente attaccabile
- Ci sono autorità fidate

Teoricamente se qualcuno ha rubato la chiave segreta del certificato (quindi l'attaccante può effettuare un MITM), un operatore della CA se ne accorge e revoca la chiave.



Ma il sistema di revoca e' facilmente attaccabile, nel mondo reale l' attaccante oltre ad eseguire un MITM tra user-agent e sito si frappone anche tra l'user-agent e il servizio OCSP.



Lo user agent non riceve risposta e considera un determinato certificato non valido: no risposta CRT non valido, mi salvo. Se siamo in questo scenario, l'attaccante è in grado di eseguire un *denial of service*. Intercetta il pacchetto e non risponde così da bloccare il servizio, il client comunica col servizio ma

non riesce a verificare se il certificato ottenuto è corretto o meno.

Devo cambiare il paradigma: quello appena mostrato è un paradigma **pull**.

L'alternativa è il **push**: non aspetto che l' user agent richieda il certificato, glielo invio subito. Appena il certificato è revocato propago l'informazione di revoca su tutti i client. Il problema di questo paradigma e' che devo memorizzare sul client tutte le revocate. Questo push è fatto dal CRL Monitor, sono mantenuti dagli stessi maintainer dei client.

Meno dura il certificato più questo e' sicuro in quanto anche se rubato il lasso di tempo per effettuare azioni malevoli e' minore.

Trust issue

Pubblica

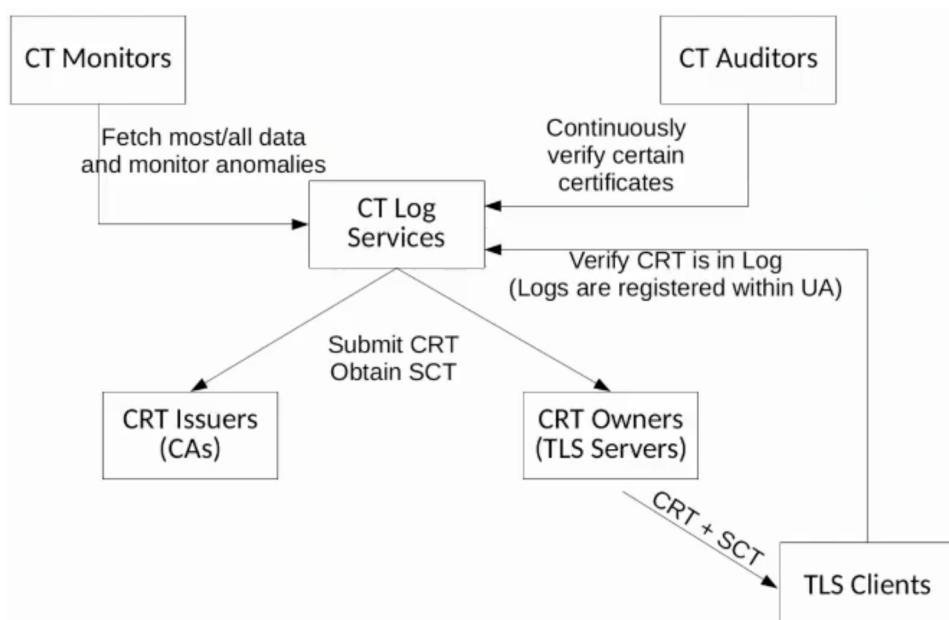
La CA pubblica e' una parte fidata, ci fidiamo che rilascia i certificati soltanto per proprietari legittimi. Come si risolve il problema del trust?

Whitelisting: creo una lista delle CA di cui mi fido, questo può essere fatto sia da utenti che dai maintainer dei software stessi.

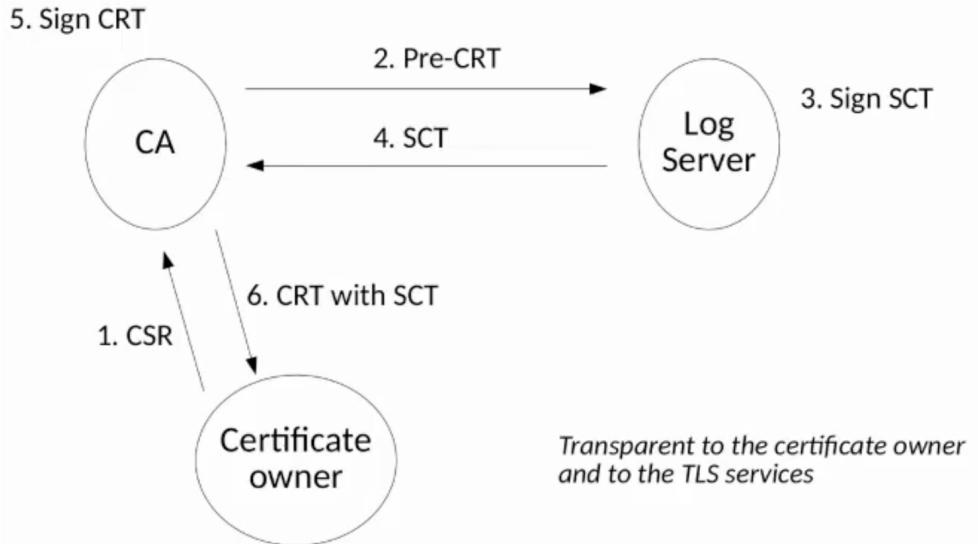
HTTP Pinning: utilizzo il paradigma TOFU, acquisisco un certificato la prima volta ma tutte le volte successive mi aspetto lo stesso certificato (poco flessibile).

Certificate Transparency (CT)

Approccio di sistema estensione della PKI stessa con servizi aggiuntivi.



Lo scopo di questa architettura e' di diminuire il potere degli attori fidati (CA), ogni certificato rilasciato deve essere inserito nel sistema di log, questi log sono pubblicamente accessibili.



In questa maniera il CA si occupa di embeddare il Signed Certificate Transparency (SCT), esistono soluzioni

- in cui si affida il compito direttamente al log server ma questa informazione non essendo più inglobata nel certificato viene distribuita tramite il primo handshake della connessione
- sfruttando OSCP quindi sempre oneroso per la CA ma più gestibile

I client (browser) richiedono informazioni aggiuntive che attestano il fatto che i certificati ricevuti siano stati salvati nei log. Per motivi di ridondanza i log vengono salvati su due sistemi.

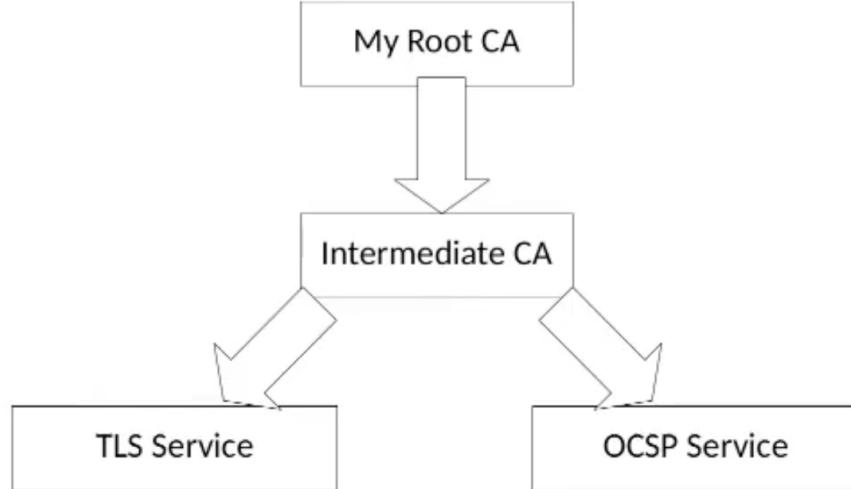
I dati non sono salvati come una lista qualsiasi ma tramite *strutture dati verificabili*, queste se il CA si ‘comporta male’ chi sta interagendo con il servizio di log se ne accorge.

Questo sistema non prevede le CA a comportarsi male ma rende tutto il processo più trasparente così da notare comportamenti malevoli.

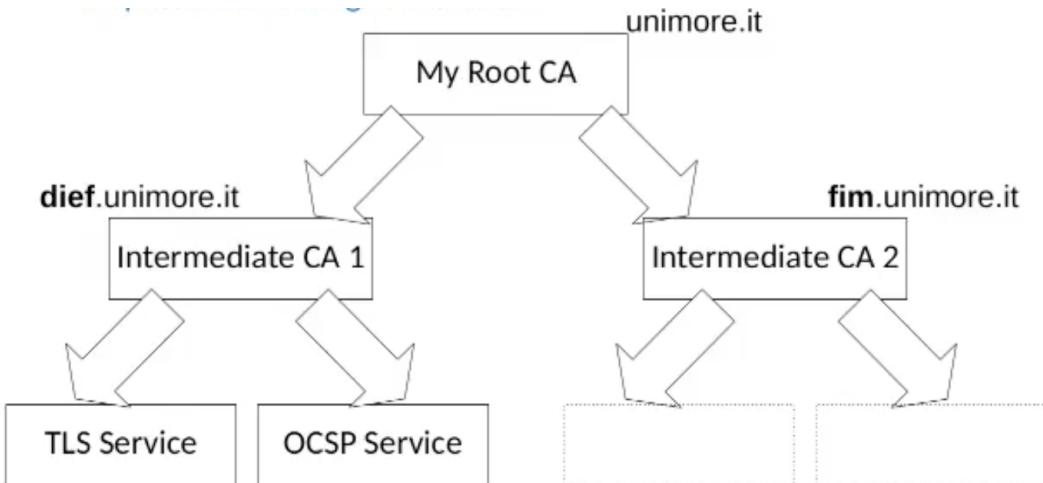
Esiste un approccio ibrido *Private CA as-a-Service* in cui ci si affida ad aziende specializzate del settore che si prendono carico di creare la CA privata, queste CA a differenza di quelle pubbliche non si appoggiano a root pubbliche ma private.

Privata

Il problema della CA privata e' sulla sicurezza, non essendo un'azienda specializzata e' facile avere vulnerabilita'.



Proprio come la struttura delle CA pubbliche solitamente viene utilizzato un approccio gerarchico, in questo modo il rilascio dei certificati avviene in maniera più sicura.



Name Constraint extension: non avere una root CA, una CA intermedia che firma tutti i certificati ma se la mia azienda ha diversi rami, posso sfruttare la gerarchia a più livelli per specializzare i certificati rilasciati da ciascuna CA.

protocollo che permette di erogare certificati automatici controllando chi ha fatto la richiesta abbia il controllo del dominio

Protocolli di comunicazione sicuri

Esistono svariati protocolli a seconda del contesto in cui li dobbiamo utilizzare.

Secure channel

Identifica il mettere in sicurezza la comunicazione tra due partecipanti in un determinato layer del network protocol stack (es. host-to-network, application, ...).

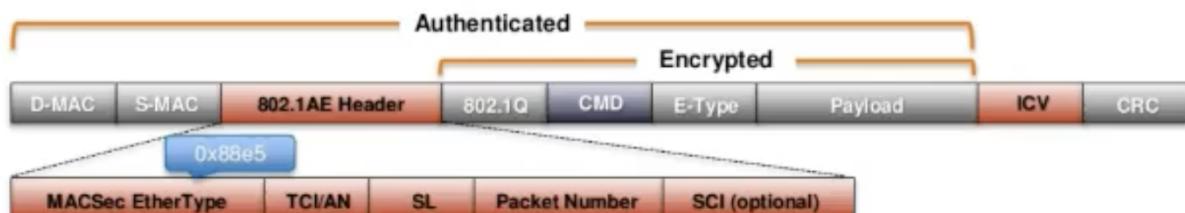
Si possono realizzare due tipi di canali:

- canale sicuro orientato al **pacchetto**, cirfo il dato e questo e' indipendente dagli altri pacchetti, proteggo la singola unità informativa senza aver cognizione del tutto
- canale sicuro orientato allo **stream**, bisogna garantire l'integrità e autenticita' di tutto lo stream

HTTP	FTP	SMTP	TELNET	DNS
TCP				UDP
IP				
Ethernet				

Tutti i protocolli nello stack TCP/IP non sono nati con l'intento di essere sicuri, bisogna estenderli per garantire la sicurezza.

MACSec



Protocollo pensato per proteggere dagli attacchi di tipo fisici (livello 2), modifica il frame ethernet aggiungendo delle tecniche per garantire autenticità e opzionalmente la confidenzialità (del payload). Essendo un protocollo molto recente e' difficile trovarlo attualmente.

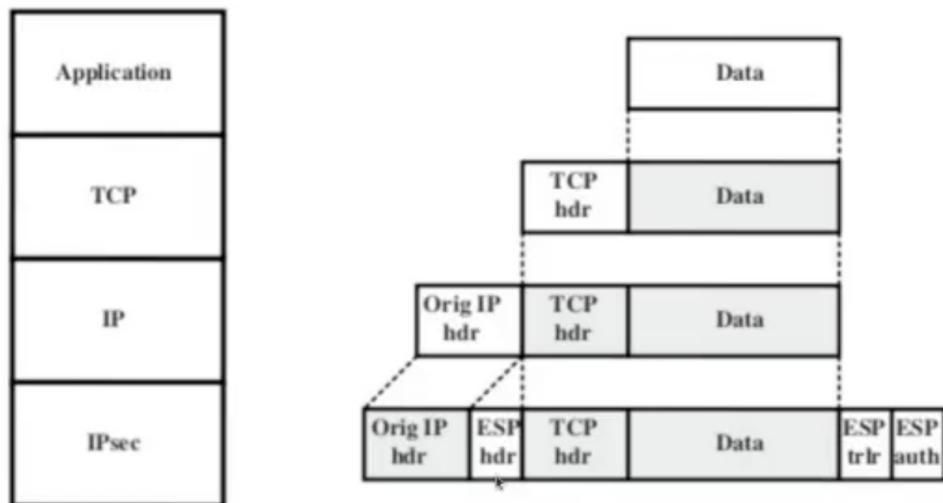
IPSec

Aggiunge la sicurezza ai pacchetti IP (livello 3), all'interno dell'header IP c'è un campo relativo al tipo di protocollo trasportato dal pacchetto, se il valore è 50 allora è IPSec.

Ci sono due modalita':

Trasporto

Sicurezza end-to-end, crea un canale di comunicazione sicuro tra i due host che vogliono comunicare e garantisce la sicurezza dei pacchetti ip

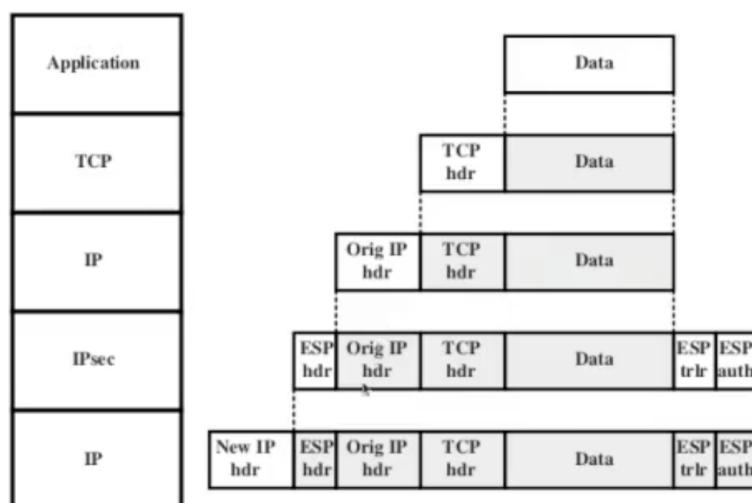


Oltre all'header ip originale si aggiunge ESP(Encapsulating security payload) Header che identifica lo schema di criptazione usato. Viene cifrato il payload e aggiunti dati come *ESP auth* ovvero il tag per garantire l'autenticita'.

L'header originale IP non viene protetto, cifra solo il payload di IP => l'attaccante può sapere tra chi sta avvenendo la comunicazione

Tunnel

Voglio garantire la sicurezza tra due reti, metto in comunicazione sicura due gateway (due reti private).



A differenza della modalita' *trasporto*, il router incapsula **tutto** il pacchetto IP compreso l'header in un nuovo pacchetto IPsec che avra' un nuovo ip (ip dell'gateway).

La rete internet non riesce a capire la comunicazione tra quali host avviene, conosce solo le due reti che stanno comunicando.

Internet Key Exchange (IKE)

Protocollo modulare pensato per IPSec per la distribuzione delle chiavi (Security Association), supporta *PKI* e certificati x509 per l'autenticità delle chiavi pubbliche. Basato su UDP sulla porta 500.

Transport Layer Security

Con il livello 4 mantengo i protocolli TCP/UDP originali e aggiungo un layer di sicurezza. Vado a creare una comunicazione sicura tra host ma in più è sicura anche a livello applicativo (utilizzo le porte).

TLS (SSL)

Protocollo di sicurezza basato su TCP che permette di creare un canale sicuro orientato agli stream.

Essendo un protocollo fondamentale è molto attaccato, le versioni attualmente sicure sono solo due 1.3 e 1.2, le versioni 1.0 e 1.1 sono state deprecate da poco (2020) in particolare la versione 1.1 utilizza protocolli sicuri ma potrebbe adoperare cifrari deprecati (es. sha1).

La versione 1.3, uscita dopo 10 anni, riprogetta il protocollo per renderlo più sicuro eliminando tutti i cifrari deprecati della versione 1.2 (sacrificando la compatibilità).

Dopo aver fatto l'handshake di TCP si esegue l'handshake di TLC che si può suddividere in quattro fasi:

1. Vengono stabilite le *security capabilities*, ovvero tutte le informazioni su come si sta stabilendo la connessione sicura (versione dei protocolli da utilizzare, cifrari, ...)
2. Il server si autentica e inizia il protocollo di scambio di chiavi (es. manda il suo certificato)
3. Client manda il suo contributo di Diffie Hellman (ed eventualmente manda i suoi certificati)
4. Fase di finish per confermare che tutta la procedura sia avvenuta correttamente, senza attacchi/compromissioni. Questi messaggi finished sono cifrati con le chiavi appena scambiate così da autenticare tutto il resto.

In TLS 1.3 hanno ottimizzato lo scambio di messaggi così da avere una fase in meno, in tal modo le performance sono migliori favorendo lo scambio anche in scenari con latenza alta.

Una funzione di derivazione di chiavi viene utilizzata per calcolare una o più chiavi crittografiche a partire da una singola informazione segreta.

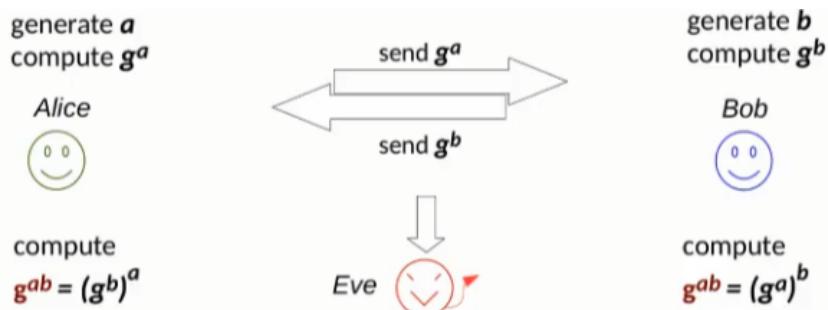
A seconda del tipo di informazione segreta esistono standard di funzioni diverse.

In questo caso da una chiave crittografica voglio ottenere molteplici chiavi crittografiche => bisogna utilizzare un approccio *hash-based* in quanto il segreto di partenza e' gia' crittografato.

Se Alice vuole accedere (oltre ai suoi dati) ai dati di Bob utilizzando però la sua chiave può dare a Bob una chiave di cifratura derivata dalla sua.

In particolare, la versione 1.2 e precedenti si reinventavano la propria versione di derivazione ma nell'ultima versione di TLS si utilizza uno standard esistente: *HKDF* (HMAC-Key Derivation Function), fatto per implementare standard di derivazione di chiavi.

Perché ci serve derivare più chiavi da una chiave master?



Queste key all'interno delle funzioni che usano alice e bob che chiave è? È g^{ab} ? È dappertutto questo? **Assolutamente no.**

E' bene che si utilizzino delle chiavi diverse, sia quando autentichiamo che cifriamo. Ci serve per questo derivare da una singola chiave più chiavi. Almeno 3, due per schemi di cifratura e schemi mac, per attacchi di reflection autenticando i dati da una direzione all'altra è bene che i dati vengano autenticati con due chiavi, per la cifratura va bene anche usare una chiave che non sia per forza diversa. La partenza è una chiave segreta, ci serve D.H. per ottenere una chiave simmetrica.

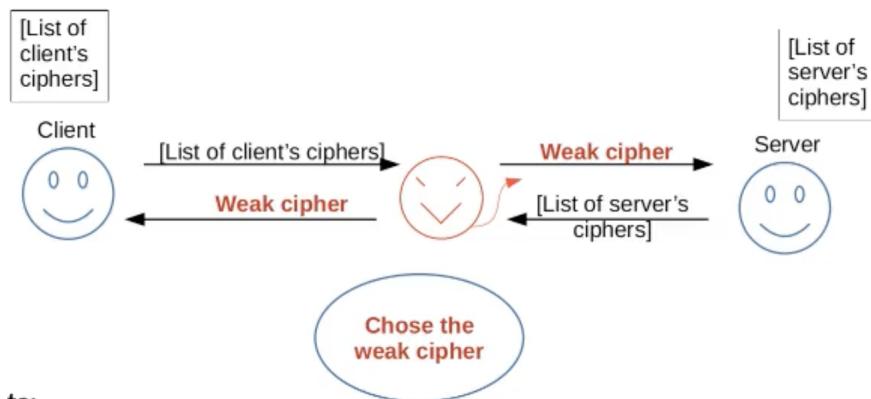
Usare una chiave pubblica sempre non è assolutamente sostenibile, lo abbiamo visto fare solamente nell'ambito delle cifrature ibride e scambio di chiavi basato su cifratura simmetrica.

Session ID: permette di applicare politiche di caching, questo *id* viene utilizzato per segnalare al server che ha già comunicato in precedenza con il client, se l'id è valido non vengono eseguite le fasi successive.

Cipher Suites: insieme dei protocolli crittografici che si utilizzano nello scambio.

Downgrade attacks

In tutti i protocolli in cui abbiamo una fase di negoziazione c'è il problema che dobbiamo scambiarsi le informazioni in chiaro (fase 1 TLS).



Server e Client si scambiano la loro lista della cipher suites, vengono scelti i metodi più sicuri in comune. Se per motivi di compatibilità inseriscono nella lista standard deprecati un possibile attaccante tramite MITM puo' far utilizzare quegli standard.

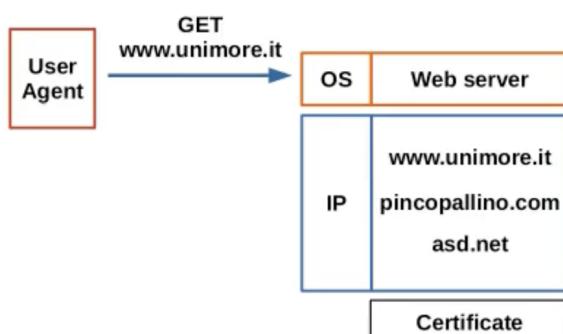
Il messaggio *finished* serve proprio per proteggersi da questi attacchi, il client quando autentica i dati non autentica il messaggio dell'attaccante ma la sua lista dei cifrari (stessa cosa fa il server), se le informazioni autenticate non combaciano con la visione precedente che hanno il client e server dell'handshake vuol dire che qualcosa e' andato storto.

TLS 1.3 c'e' una modalità *0-RTT* dove non viene fatto alcuno handshake e viene utilizzata quando si ha la sessione cachata, di default e' disattivata ma permette di avere performance elevate, per natura questo tipo di comunicazione non può essere protetta dal reply attack.

DTLS

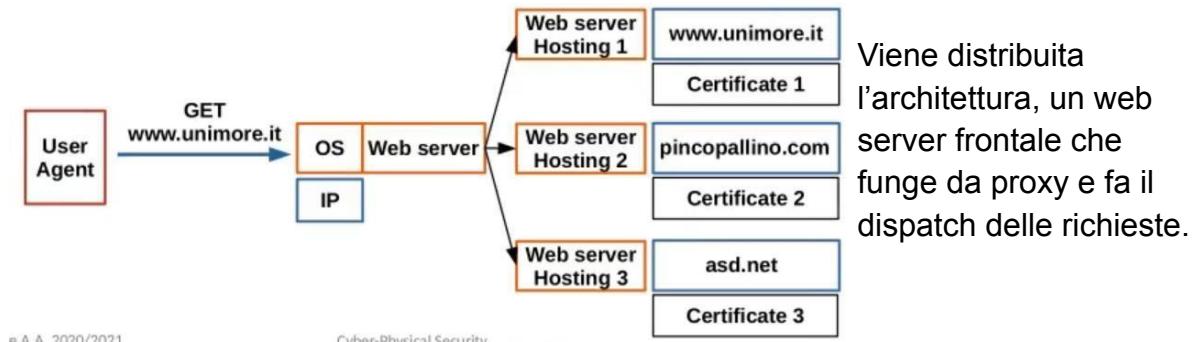
Protocollo basato su TLS ma essendo orientato ai pacchetti non garantisce la protezione dal riordino, ogni pacchetto e' indipendente dall'altro. Nonostante sia costruito su UDP richiede l'handshake.

Virtual Hosting



Serve per fare multiplexing di più siti sullo stesso web server, quando si fa la richiesta http si specifica nel campo *host* quale sito si vuole raggiungere. Questa non e' una struttura performante.

La gestione dei siti dell'internet moderna e' più simile a questa:



Come si può comunicare al web server come effettuare la distribuzione delle richieste se le informazioni sono cifrate? L'hostname e' all'interno del pacchetto cifrato sulla base dello specifico certificato.

In una connessione HTTPS per gestire correttamente il virtual hosting di default il client inserisce nell'handshake TLS un'informazione in chiaro: il nome dell'host.

SSL Stripping



Approfitto che molti client contattano il web server in http per fare un MITM, l'attaccante si finge il server con la vittima e nel frattempo stabilisce una connessione sicura con il server originale.



Unico modo per evitare questo attacco e' dire al client di non utilizzare MAI http, neanche per la prima richiesta.

HSTS è un protocollo che serve per settare un Strict Transport Security che, tramite l'header di risposta, impone di usare https dalla seconda connessione in poi (simile all'approccio TOFU). Inoltre ci sono dei crawler che potrebbero collezionare in

anticipo quali sono i server che utilizzano questo tipo di risposta per inserirli direttamente all'interno dei browser (<https://htstspreload.org>).

SSH

Protocollo applicativo per connettersi a terminali remoti, non basato su TLS ma ha un suo protocollo di sicurezza.

SSH User Authentication Protocol Authenticates	SSH Connection Protocol Multiplexes the encrypted tunnel into several logical channels
	SSH Transport Layer Protocol Provides server authentication, and data confidentiality and authenticity. It may optionally also provide compression
	TCP

A livello di sicurezza ha un layer simile al TLS, sopra a questo protocollo ha due ulteriori protocollli: uno adibito all'autenticazione e l'altro permette di effettuare un multiplexing, più connessioni all'interno dello stesso layer di trasporto.

Identity file: contengono coppia di chiavi asimmetriche che il client/server utilizza per lo scambio di chiavi con Diffie-hellman, quindi molto simili ai certificati ma non e' pensato per l'utilizzo globale quindi non sono presenti meta dati.

Sistema di autenticazione di default e' la password, un metodo più sicuro e' autenticarsi tramite la chiave pubblica. Per farlo bisogna inserire la propria *public key* sul server nell'utente in cui si vuole accedere (cartella .ssh/authorized_keys).

```
Host 192.168.56.104
  Hostname 192.168.56.104
  User user
  IdentityFile ~/.ssh/mykeys/id_test
```

Se si gestiscono connessioni ssh a più server e' possibile modificare il file di configurazione in .ssh/config , qui si può specificare a seconda dell'host quale chiave utilizzare e con quale utente accedere.

Secure Email

Nonostante sia un mezzo 'vecchio' per comunicare, rimane l'applicazione più utilizzata per scambio di messaggi tra aziende o da aziende a clienti.

Phishing

Attacco in cui il destinatario riceve una comunicazione che rimanda su un sito internet dove deve accedere e inserire le sue credenziali, la mail e' falsa, il sito e' falso. L'obiettivo dell'attaccante e' rubare le credenziali.

Per difendersi dal *phishing* ci sono due approcci principali:

- essere stati educati a riconoscere un'email falsa
- utilizzare protocolli avanzati di autenticazione *WebAuthn*, il protocollo riconosce il sito falso e le credenziali non vengono rubate

Malware

L'utente riceve un allegato, lo scarica e lo esegue.

È un programma che quando è eseguito sul computer compromette il sistema, facendo dei danni di diversa natura.

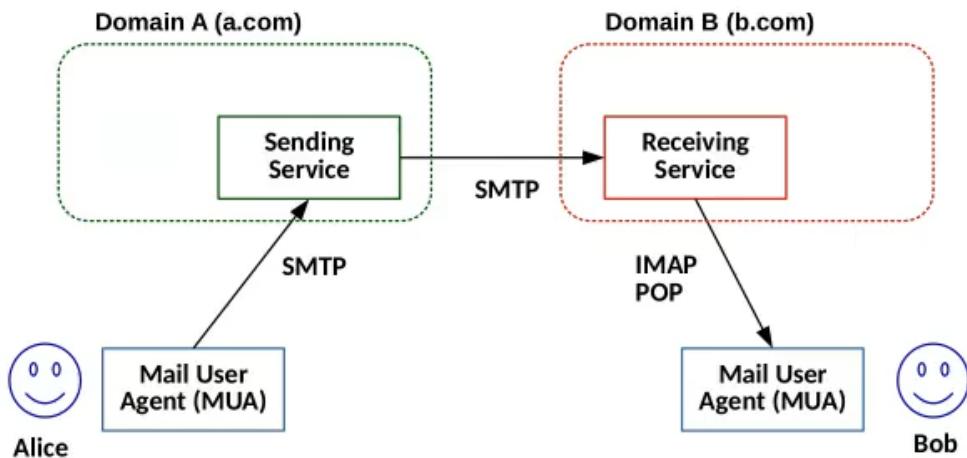
Il più famoso attualmente è il *ransomware*, questi cifrano i file sul computer e chiedono un pagamento per avere la chiave di decifratura.

Si possono ancora mandare eseguibili via mail? Google lo nega assolutamente. A volte questi malware possono essere dei file che sembrano legittimi ma sfruttano delle vulnerabilità dei programmi con cui vengono aperti.

Spoofing

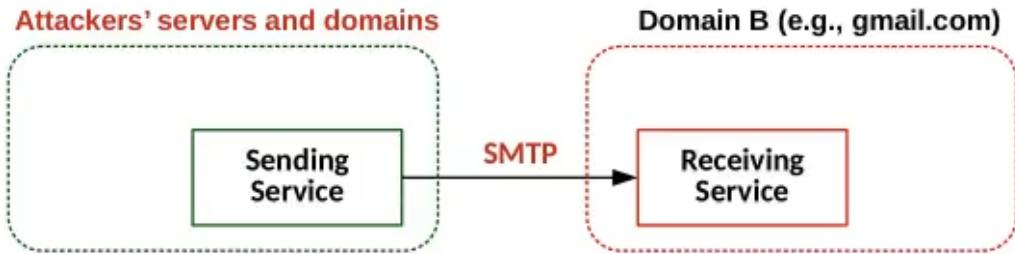
Attacco che vale per email ma anche per qualsiasi tipo di autenticazione, un attaccante invia un messaggio cercando di falsificare il mittente.

Funzionamento email

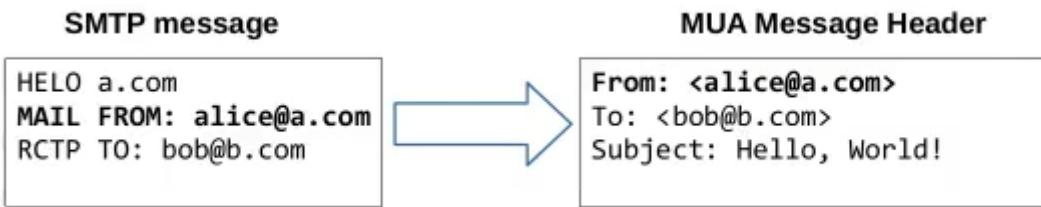


Sistema **distribuito**, non c'è un'entità unica responsabile di tutti gli utenti che possono inviarsi email. Alice invia un'email al server di dominio (tramite SMTP) di Bob.

In tutta questa transizione e' difficile per il dominio destinatario capire se una email e' di un utente legittimo.



L'identità non è un'informazione contenuta in qualche campo che viene verificato che è soggetto ad un controllo di autenticazione forte, ma viene inserita nel corpo dell'email stessa.



Un messaggio SMTP autodefinisce qual'è il mittente della email. Non c'è modo di verificare il campo MAIL FROM.

Per evitare il problema dello *spoofing* sono state introdotte delle misure di sicurezza. Lato provider(google, libero, aruba...):

- Sender Policy Framework (SPF)
- DomainKeys Identified Mail (DKIM)
- Domain-based Message Authentication, Reporting & Conformance (DMARC)
- Brand Indicators for Mes-sage Identification (BIMI)
- Authenticated Received Chain (ARC)

Lato utente (end to end) per proteggere l'autenticità e la confidenzialità dei messaggi:

- S/MIME
- PGP

Email security and legal value in Italy: PEC

Il receiving service prima di accettare la mail e inviarla all'user-agent del destinatario applica dei controlli, utilizzando protocolli **SPF** e **DKIM**.

In particolare *SPF* consulta il server DNS che corrisponde al dominio del mittente e fa una query per record chiamati TXT (record generici) in cui dovremmo trovare l'elenco degli indirizzi IP dei servizi autorizzati a inviare email così da verificare che l'email sia stata inviata da un provider autentico (e non da uno che si finge ad esempio Alice mail).

DKIM, a differenza di SPF, non si basa sull'indirizzo ip sorgente ma sulla firma del *sending service* ed in modo simile a SPF fa un lookup DNS per cercare quali sono le chiavi pubbliche valide del servizio in modo da verificare le firme ricevute dal sending service.

Utilizzare questi protocolli e' a discrezione del provider, un utente a seconda del provider email e' piu o meno vulnerabile. Anche chi li implementa li possono utilizzare con scelte finali differenti: ad esempio rifiutare una email se questa non utilizza protocolli di sicurezza (mandata da un provider che non firma la mail, non usa i record DNS, ...), oppure accettarle tutte e dare un punteggio di spam alle email.

Il provider può verificare il servizio da cui arriva l'email(es. gmail) ma non può controllare il mittente di questa (es alice@gmail.com).

S/MIME (Secure MIME)

Standard per sicurezza end to end che si basa su PKI, per firmare e cifrare le email vado ad utilizzare delle chiavi salvate in dei certificati x509 .

Mittente invia una email, la firma utilizzando un certificato, il destinatario riceve il certificato controlla che questo risalga ad una Root Certification Authorities e verifica l'autenticità.

PEC

Subset di S/MIME (non ha cifratura) rilascia dei certificati del tutto simili a PKI ma che sono controllate a livello italiano e sono predisposte a controllare l'identità legale che si lega direttamente con te. Questo serve per sfruttare il concetto di non ripudiabilità, se firmo una e-mail non posso dire che non sia mia, perché sono l'unico detentore della chiave per firmarla.

E' presente anche il concetto della ricevuta di ritorno, firmata e mandata dal servizio ricevente. Inoltre il ricevente e' obbligato per legge a leggere la PEC.
Questo sistema e' esclusivamente italiano.

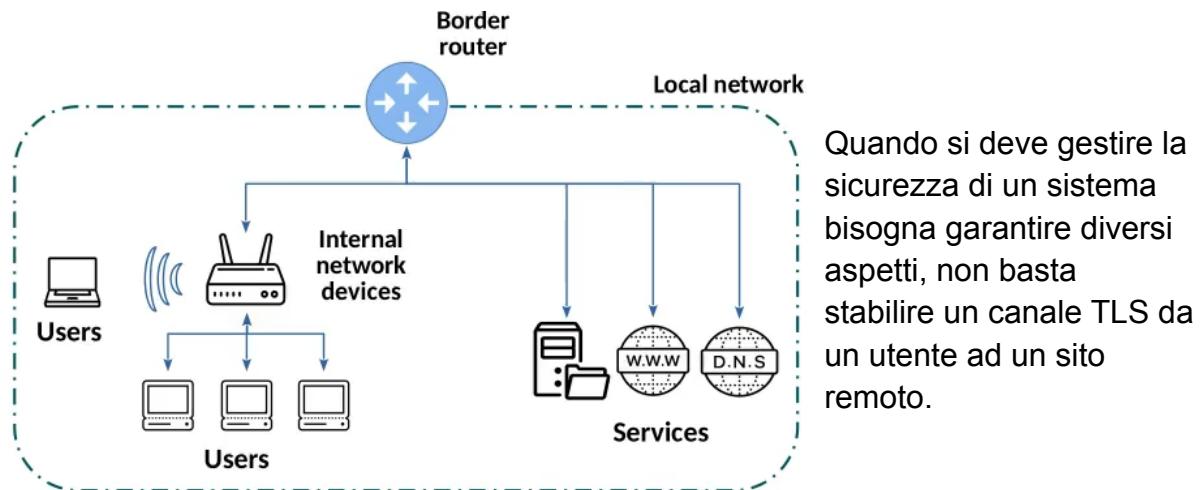
PGP

Protocollo che firma e cifra le mail come S/MIME ma scollegato da PKI, ha un ottica più decentralizzata. GPG software famoso per utilizzare il framework OpenPGP (specifica implementazione del protocollo PGP).

Configurazione di default: una sola chiave per firmare e per cifrare i messaggi.

Esattamente come in PKI si puo creare un'infrastruttura intermedia per aumentare la sicurezza: si ha una chiave master (che teniamo al sicuro) ma ogni giorno si utilizzano le *subkey* ovvero sottochiavi con ruoli specifici (es. una per firmare e una per cifrare) così da poterle revocare (tramite *master key*) nel caso di perdita/furto.

Sicurezza di rete



In una rete locali sono presenti:

- Nodi (notebook, switch, router ...) gestiti da utenti o servizi (webserver)

Gli attaccanti non attaccano solo la tecnologia (violare una chiave...) ma anche violare persone che hanno accesso al sistema.

Attack surface: L'attaccante può aver accesso al sistema da diversi punti.

Attaccare le persone vuol dire convincerle a fare qualcosa che non dovrebbero, l'attacco più popolare e' mandare comunicazioni di *phishing*:

- mass/spam phishing: comunicazioni/email mandate a moltissimi utenti ma facilmente riconoscibili a causa della loro generalità'
- spear phishing: attacca una specifica persona, solitamente include informazioni personali

Social engineering: ottenere informazioni sulle persone per rendere l'attacco più realistico quindi avere maggior probabilità di successo. Un attacco popolare e' una situazione in cui c'è un rapporto di fornitura tra due aziende => email di cambio IBAN

Come difendersi? Educare le persone, renderle a conoscenza di questi attacchi.

Principi di sicurezza

Defensive-in-depth

Quando si realizza la sicurezza di un sistema non bisogna creare un approccio alla sicurezza a singolo livello, ci devono essere più strati di sicurezza, se un livello fallisce non deve collassare tutto.

Separation and Segregation

Devo fare in modo che tutte le mie risorse siano separate, nel momento in cui queste partizioni vengano messe in contatto io possa effettuare controlli.

Least privilege principle

Dare i minimi privilegi possibili per eseguire quel determinato task.

Sicurezza dei servizi

Se un attaccante prova a violare dei servizi di rete in esecuzione sulla rete, solitamente proverà a sfruttare delle specificità di quel servizio ma prima di farlo c'è una fase comune: **Network reconnaissance**, l'attaccante investiga sulla rete cominciando ad analizzare servizi noti (pubblici), sfrutta informazioni laterali.

Attraverso l'**address sweeping** si evidenziano quali hosts sono attualmente attivi (i potenziali target). Ottenuti i target, tramite **port scanning** si controllano le porte aperte. Infine si cerca effettivamente una identificazione: **Service Fingerprinting**, capire qual è l'applicativo in esecuzione, informazioni circa i dettagli tecnici del servizio vittima .

Posso anche però avere un attaccante locale, un impiegato che si è stufato e decide di attaccare la sua stessa azienda; che cosa può fare un soggetto del genere?

Magari se è tutto ben segregato può avere solamente informazioni del suo subset di riferimento.

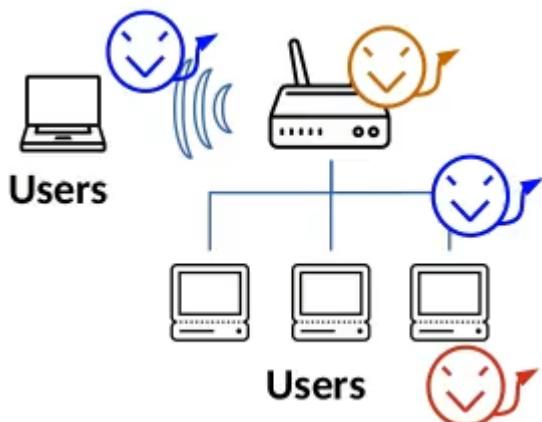
A seconda dell'attacco ho soluzioni tecniche diverse (Segmentation e Segregation a diversi livelli dello stack di protocolli):

- **VLANs:** segmentazione e segregazione a livello 2 (per evitare attacchi di livello 2);
- **L3 e L4 Firewalls:** sistemi che vogliono effettuare operazioni di filtraggio e controllo a livello di IP e di trasporto;
- **Deep Packet Inspection (DPI):** è sempre un firewall, ma ha la capacità di ispezionare i pacchetti anche in profondità, dove questo vuol dire andare nei livelli alti dei protocolli (può capire informazioni Header HTTP o altri specifici);

- Application Layer Firewall: strumenti di segmentazione e segregazione appositamente pensati per certi applicativi di rete;

Operazioni di *netting*, possono andare a nascondere IP privati all'interno di una rete, rispetto ad una rete pubblica. Questo è un concetto di segregazione e segmentazione, creo una rete di indirizzi non indirizzabili all'interno della rete stessa.

Attacchi al Livello 2



Se un attaccante riesce ad attaccare la rete a livello 2 => ha accesso alla rete locale.

Bisogna separare e segregare, far in modo che le reti locali siano ben separate, ognuna deve avere una competenza specifica sia per migliorare le performance ma soprattutto per la sicurezza. Se ad esempio nella stessa stanza ci sono persone con ruoli differenti e privilegi differenti, installare uno switch per ogni ruolo non è una buona soluzione, per

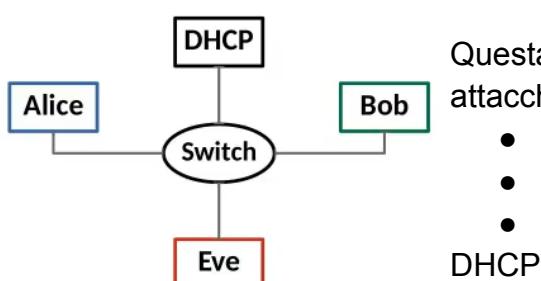
creare logiche di separazione e segregazione in dispositivi di livello 2 si utilizzano le **VLAN** (estensione del protocollo ethernet).

Hub: dispositivo di rete che riceve un frame ethernet da una porta e lo forworda in tutte le altre, stesso dominio di collisione e broadcast (inoltro di massa).

Switch: avendo il dominio di collisione separato riesce ad inoltrare a porte specifiche i messaggi unicast però rimane il fatto che i messaggi di broadcast deve inviarli a tutte le porte e questo può essere una falla.

MAC si basa sul protocollo ARP e si basa su due tipi di messaggi:

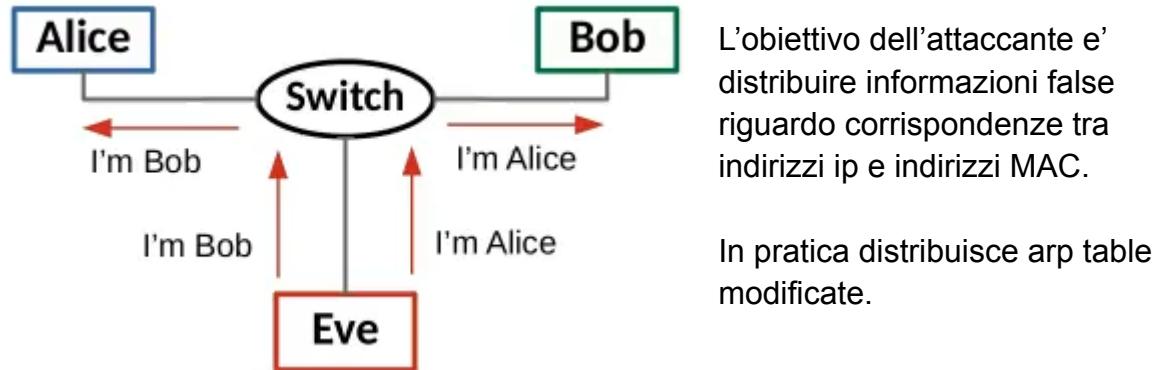
- ARP Request: messaggio broadcast
- ARP Replay: unicast message



Questa è la struttura di rete di riferimento per gli attacchi che verranno approfonditi:

- Arp spoofing => attacco ad host
- Port stealing => attacco allo switch
- DHCP poisoning => attacco al protocollo DHCP

ARP spoofing



ettercap -T -M arp /192.168.1.1//

1. Ispeziona come e' fatta la rete locale, quali indirizzi ip sono assegnati
2. Ping degli host trovati
3. Invio delle risposte arp (*arp reply*) con informazioni false agli host

L'operazione selettiva dello switch va a favore dell'attaccante, l'*arp reply* che manda Eve arriva solo ai diretti interessati.

Questo attacco e' invisibile all'utente 'normale' ma un amministratore di rete e' in grado di rilevarlo abbastanza facilmente.

Per contrastare questo attacco si puo':

- popolare la tabella arp non come cache ma come una tabella permanente (statica)
- approccio di sistema segregando meglio l'accesso allo switch stesso

Port stealing

Non vengono attaccati gli host ma lo switch.

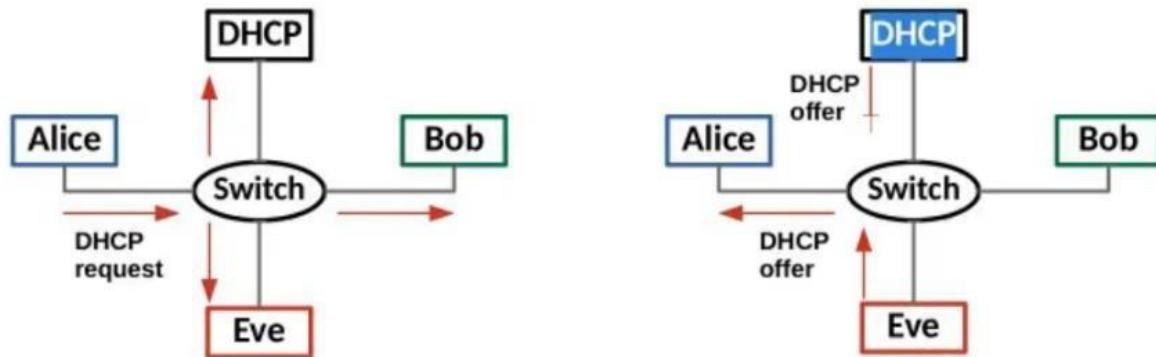
Effettuare sempre un operazione di spoofing degli indirizzi MAC ma rispetto allo switch.

Eve manda messaggi arp allo switch per far credere che sia sotto controllo degli host Alice e Bob.

Questo attacco e' piu debole in quanto facilmente rilevabile e difficilmente mantenibile dall'attaccante in quanto ogni volta che Alice/Bob manda un messaggio lo switch cambia l'associazione tra MAC:IP.

DHCP poisoning

Anche qui si tratta di attaccare un meccanismo di autoconfigurazione.



Quando Alice si connette per la prima volta alla rete manda una richiesta DHCP (DHCP request) questa richiesta e' broadcast in quanto non sa l'ip del server DHCP => Eve anticipa il server DHCP server mandando una risposta (DHCP offer) con quelle che dovrebbero essere le informazioni di configurazione di Alice (ip, regole di routing, ...).

Alice dopo aver mandato la request l'unica risposta (offer) che considera valida e' la prima che arriva. Nell'DHCP offer deve essere presente un *xid*, un identificativo (inserito inizialmente nella DHCP discover) che non permette ad un ipotetico attaccante uno spam continuo di DHCP offer, obbligandolo a ricevere prima la discover per poi tentare di anticipare il server DHCP.

L'attaccante può fingersi il gateway della rete oppure effettuare *DNS Poisoning*.

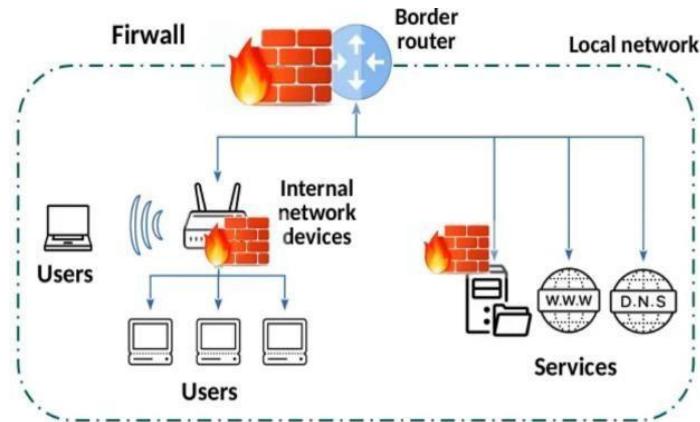
Esempio DNS Poisoning: Alice vuole pingare Bob, manda un messaggio al server DNS (non legittimo, comunicato nell'offer) così Eve può rispondere per conto di Bob o qualunque altro utente con cui Alice voglia comunicare.

Firewall (Livello 3/4)

strumenti che servono a creare regole di segregazione, devo controllare a livello 3 come possono comunicare le diverse reti.

Principi fondamentali:

- L'efficacia dipende strettamente dalla configurazione, non e' uno strumento 0 o 1 (non basta installarlo).
- Devono essere a loro volta molto sicuri in quanto possono essere target di attacchi
- Bisogna posizionarli correttamente nella rete



Si possono attuare due approcci principali:

- **implicit negation:** blocco tutto, definisco tramite white list le regole di traffico consentiti => security over usability
- **implicit allow:** consento tutto, se c'è qualcosa di strano blocco quella specifica sorgente => usability over security

Posso usare delle policy di negation o allow a seconda del tipo di traffico che vado a gestire, es. in entrata implicit negation ma implicit allow in uscita.

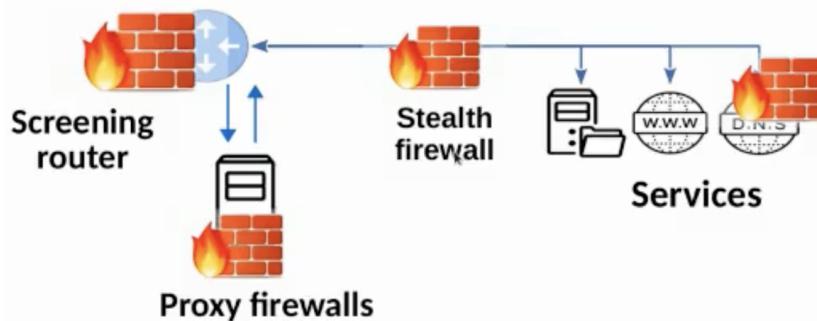
Possiamo usare i firewalls come sistema software ma anche come hardware perché ci vengono vendute delle soluzioni, ad esempio i router, con delle funzioni di firewall implementate. L'utilità in questo caso è di solito quella di avere un sistema ad alte performance, progetto hardware apposito per avere anche l'applicazione di regole di firewall.

Application gateway: firewall software/dispositivi che controllano specificatamente il traffico di certe applicazioni.

Differenti tipi di analisi sui pacchetti che possono essere effettuati:

- Static/Stateless: ogni pacchetto viene analizzato in maniera indipendente
 - più semplice e veloce
- Stateful: si analizzano flussi di traffico, gruppi di pacchetti, es. riuscire a capire il concetto di connessione, quali pacchetti fanno parte di una stessa sessione HTTP
 - oneroso a livello di tempo di computazione

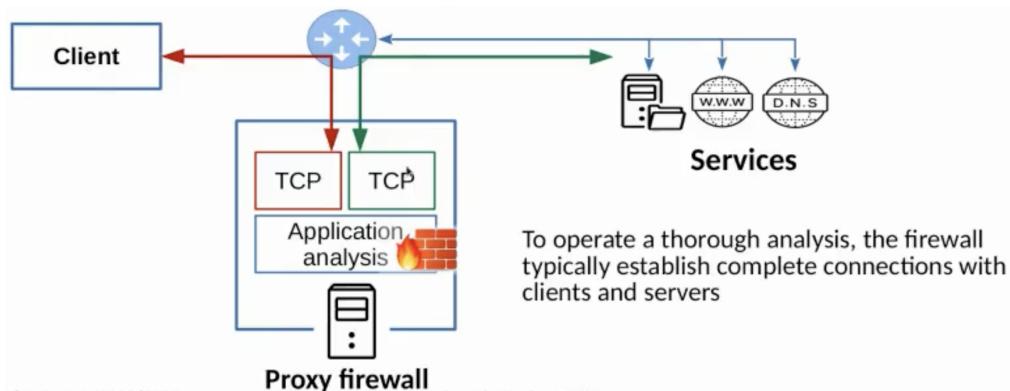
Nell'ambito del deployment di rete ci sono diversi tipi di firewall



Screening router: un router che effettua un controllo del traffico.

Stealth firewall: firewall difficilmente rilevabile all'interno della rete, a livello di funzionalità simile allo *screening router*.

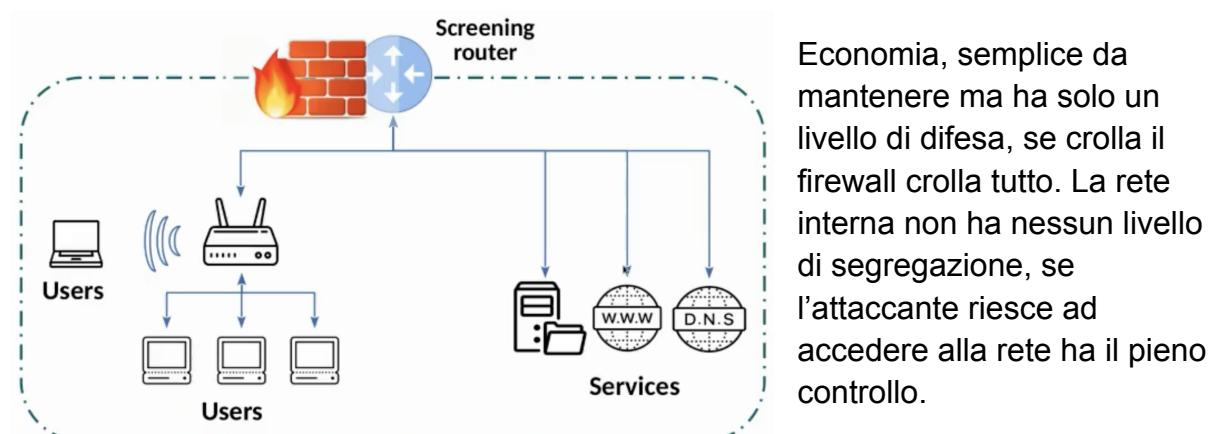
Proxy firewall: al contrario dei *stealth firewall* sono rilevabili, analizzano il traffico in transito ma implementando un analisi dei protocolli applicativi.



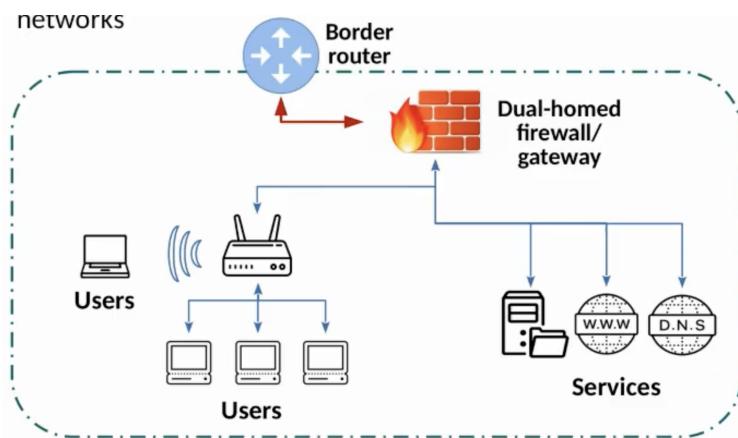
Per riuscire ad analizzare al 100% il livello applicativo spesso è necessario creare delle connessioni esplicite tra il client e il server che si ospita (un MITM a scopo di sicurezza). Quando questi firewall sono trasparenti (non vengono rilevati dal client) si chiamano *transparent proxy firewall*. Costoso a livello di performance.

Tutte queste soluzioni le possiamo usare in parallelo, una non esclude l'altra.

Single screening router

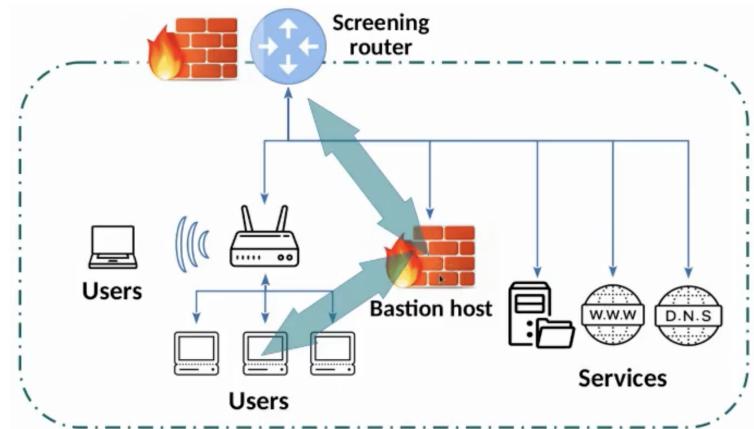


Dual-homed firewall/gateway



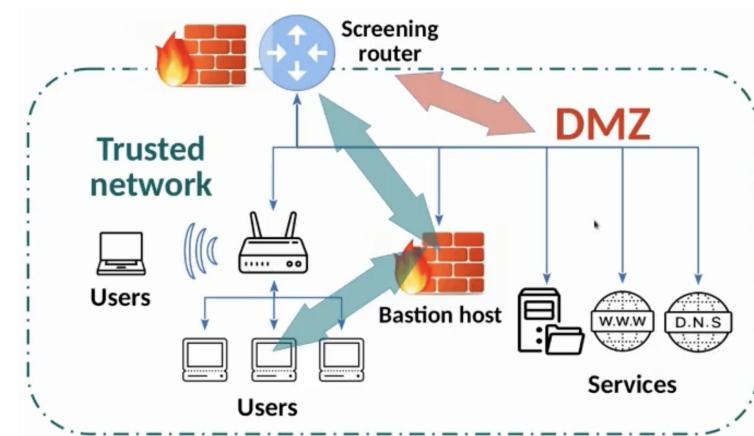
Ci sono due interfacce separate, divisione netta tra la parte esterna (untrusted) e la rete interna (trusted). Con dispositivi separati si ha un vantaggio a livello di performance.

Bastion host



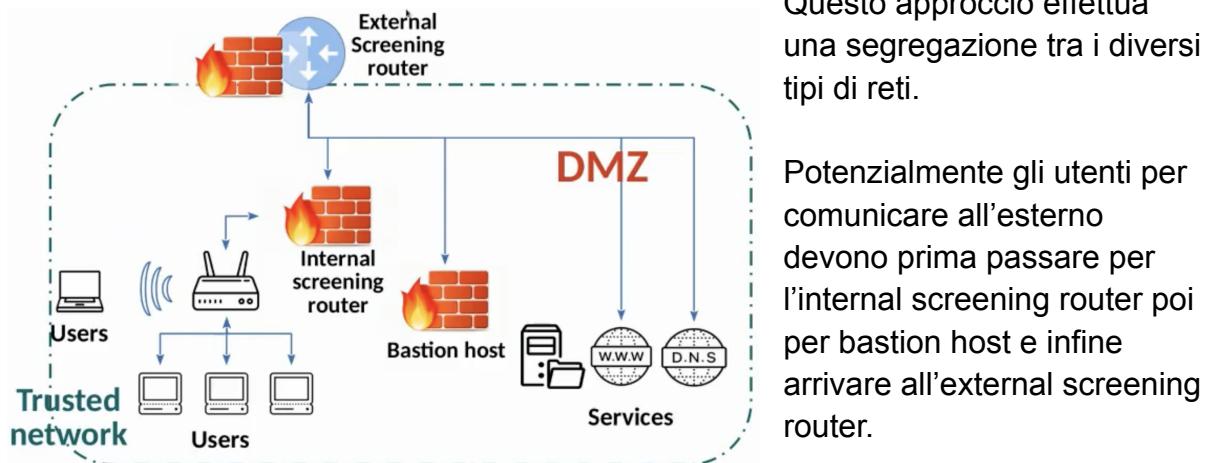
Proxy firewall intermedio, si obbligano tutti i client della rete a passare dal *bastion host*, lo screening router accetta solo pacchetti provenienti dal *bastion host*. Non offre una separazione forte delle reti, non ha funzionalità di routing, lavora a livello 5 (applicativo).

De-Militarized Zones (DMZ)



Riuscire ad avere una parte della rete che comunica con l'esterno a livelli di sicurezza inferiori.

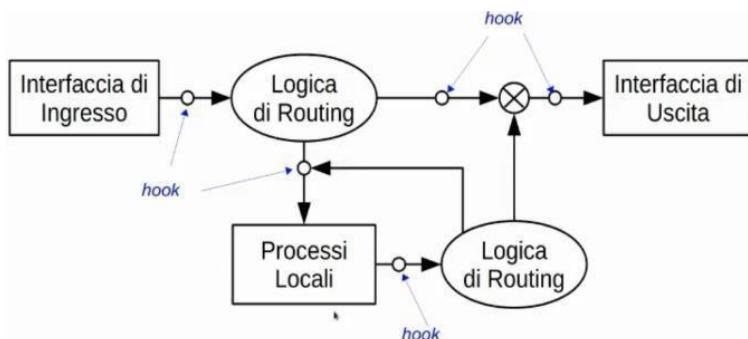
Ad esempio voglio che gli utenti della rete passino per il bastion host ma il mio webserver no in quanto svantaggioso a livello di performance.



L'host collegato alla DMZ solitamente è maggiormente protetto a livello di configurazione della macchina in quanto non c'è un firewall intermedio che lo protegge.

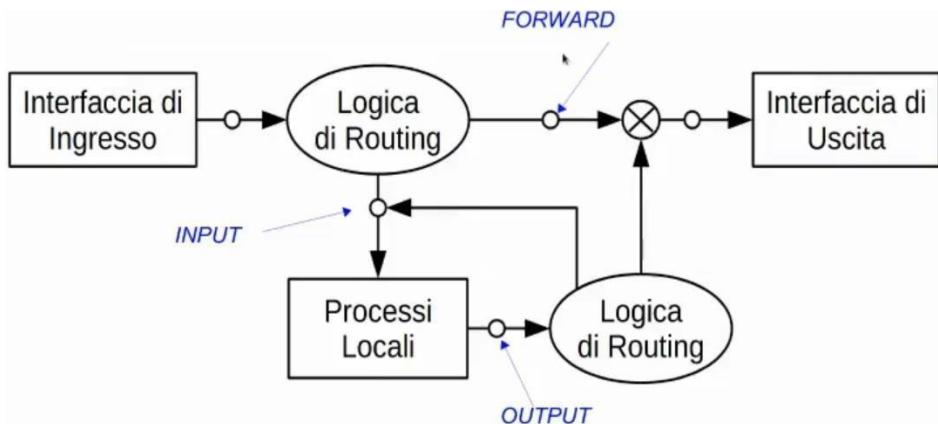
IPTables

Tool su sistemi linux che serve per fare operazioni di NAT, packet filtering e marking dei pacchetti. Serve per fare analisi, segregazione e trasformazione dei pacchetti in transito. Questo schema



concettualmente rappresenta la logica dei pacchetti in un dispositivo di rete, diversi percorsi che i pacchetti possono prendere. Il dispositivo di rete applica la Logica di Routing e cerca di capire dove deve indirizzare il traffico.

Le operazioni che ci interessano studiare di questo tool sono quelle di Filtering per vedere quale traffico accettiamo a quale no.



Input: ingresso nei processi locali

Output: in uscita dai processi locali

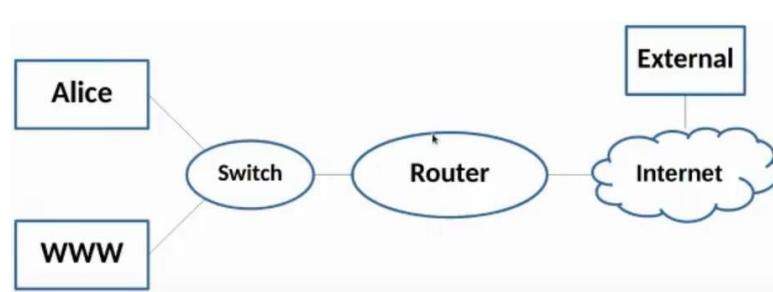
Forward: specifico per l'inoltro da un'interfaccia all'altra

In Input possiamo fare delle regole di accesso, per gestire il traffico verso i processi locali. Output: controlla le regole di accesso per il traffico in uscita dai processi locali. Forward controlla le regole di accesso per traffico in transito tra due interfacce di rete.

Chiaramente questo tipo di approccio è applicabile a qualsiasi tool per la creazione di un firewall.

Per regole intendiamo che su IPTables ci siano delle tabelle che siano insieme di regole che ci servono per uno stesso scopo. In pratica troviamo 3 tabelle di default: 2 per il filtraggio e 1 per il NAT.

Le catene sono insieme di regole e identificano tutte le regole applicate sullo stesso punto di analisi del traffico. La *tabella* ha tutte le regole nell'ambito della funzionalità ad alto livello del filtering.



La *catena* dice invece tutte le regole sul traffico che finisce in INPUT, per esempio. Una catena si chiama così perché è un insieme di regole che vengono inserite in modo sequenziale. Ogni catena non ha un insieme di regole

sparso, ma tutte le regole vengono controllate una ad una in maniera rigorosa.

Concettualmente la rete è fatta di modo che si abbia la rete privata, composta da Alice, WWW e router, quest'ultimo è collegato a Internet e poi a External.

Obiettivo e' creare sempre delle regole precise, se controllo troppo il traffico e metto regole troppo stringenti, finisce che poi gli utenti non riescono a fare nulla.