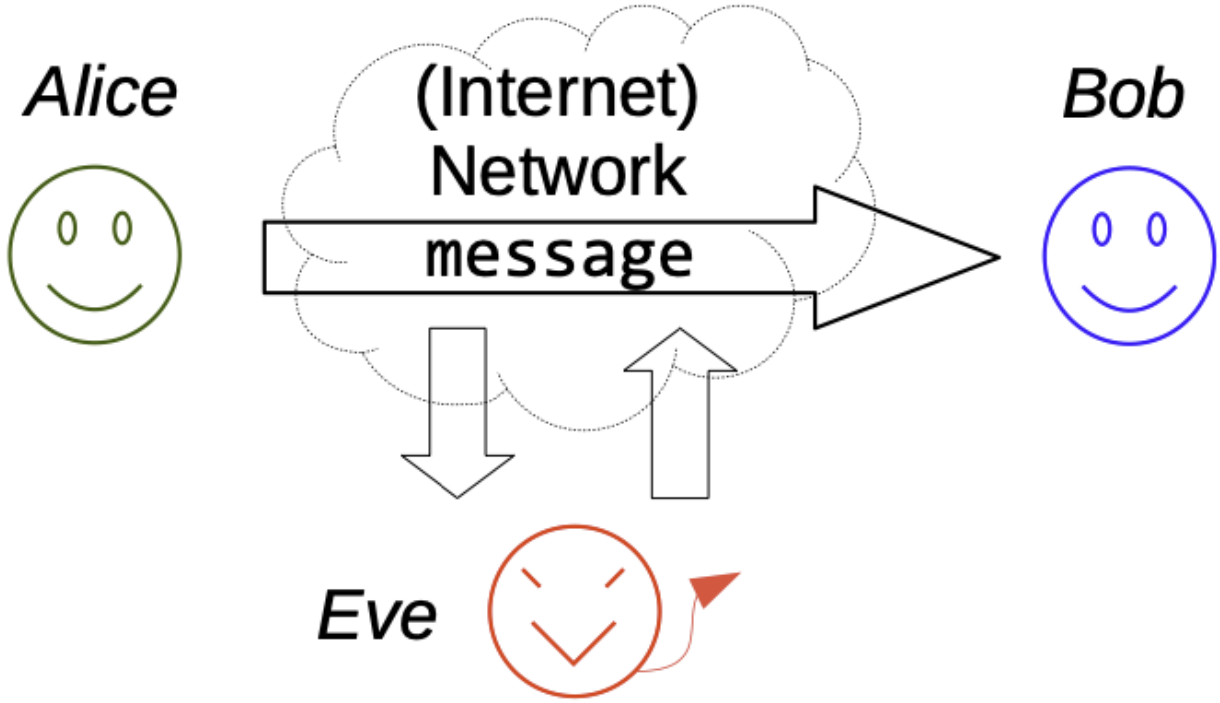
La sicurezza è un aspetto fondamentale delle reti di oggi --> garantire la sicurezza vuol dire proteggere i nostri sistemi da persone malevole.

Un protocollo sicuro deve implementare almeno queste caratteristiche:

- confidenzialità dei dati in transito
- autenticità dei dati



Concetti di protocolli sicuri

La cosa più semplice è la **secure envelopes**. La crittografia è la disciplina che permette di proteggere dati cifrandoli. I termini principali sono:

- **encrypt** (cifratura)
- **decrypt** (decifratura)

Se si cifra un messaggio si ottiene un *ciphertext*.

Le caratteristiche principali che un protocollo sicuro deve proteggere sono:

- **CONFIDENZIALITÀ** --> i dati in transito non devono poter essere "letti" da agenti esterni (eve non deve leggere il messaggio)
- **INTEGRITÀ** --> il destinatario della comunicazione riescono ad accorgersi se il messaggio è stato modificato nell'invio
- **AUTENTICITÀ** --> il destinatario può verificare se il messaggio è arrivato veramente dal mittente

> La Best Practice è cercare di proteggere tutto.

Quando si parla di crittografia si parla di:

- **Settings** --> scenario che fa riferimento a un modello di comunicazione, e può essere **Simmetrico** o **Asimmetrico**

In generale quando si cifra un messaggio si utilizza una **chiave di cifratura**

Simmetrico

Scenario che prevede l'utilizzo della stessa chiave di cifratura sia per cifrare che per decifrare

Asimmetrico

Scenario che prevede l'utilizzo di due chiavi differenti per la cifratura e la decifratura

---

Scambio delle chiavi

La prima operazione quando dobbiamo usare un protocollo sicuro è capire qual'è il modello che si adatta meglio allo scenario --> c'è anche la possibilità che si utilizzino più protocolli

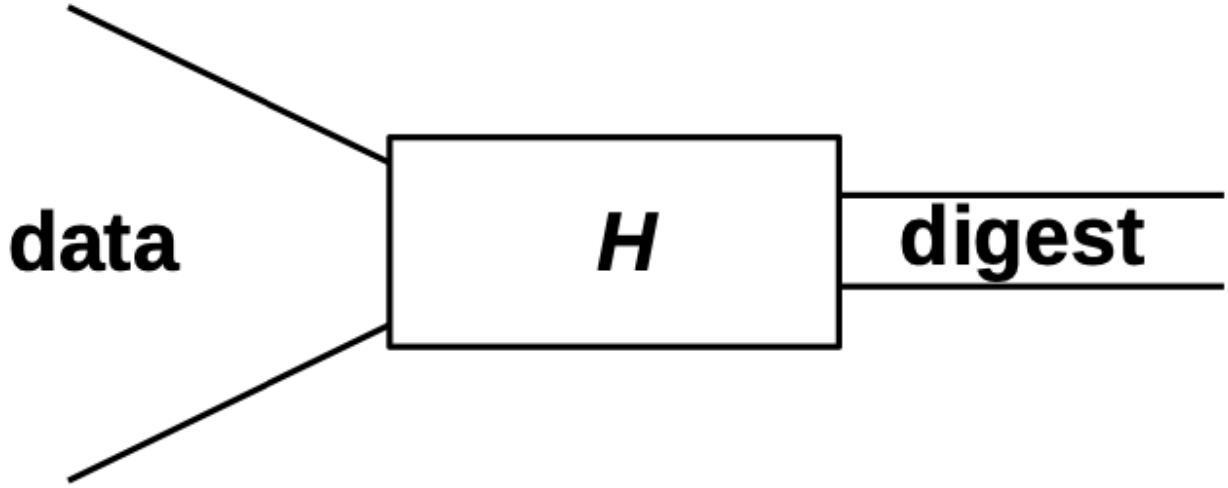
termini tecnici

Uno schema di cifratura è composto da almeno

- `keygen([size]) → key` --> funzione che genera una o una coppia di chiavi
- `encrypt([message], key) → ciphertext` --> funzione che preso un messaggio lo cifra con la chiave
- `decrypt([ciphertext], key) → message` --> funzione che preso un testo cifrato lo decifra con la chiave

Funzioni HASH

Le funzioni HASH crittografiche sono funzioni che mappano un dato di arbitraria lunghezza in stringhe di lunghezza fissa. Le funzioni HASH che si utilizzano in crittografia si dicono *resistenti alle collisioni*, ovvero si assume che dati due input differenti non esistano mai due **digest** uguali.



Questo tipo di funzioni ci permette di soddisfare il vincolo di **Integrità**, infatti si può riconoscere se il messaggio è stato modificato. L'esempio più banale è quello nell'utilizzo nei mirror.

Anche se permette di capire se il dato è stato modificato, non esiste il concetto di **data origin**. L'uso di una funzione di hash non è sicuro da solo come sistema di sicurezza.

Standard

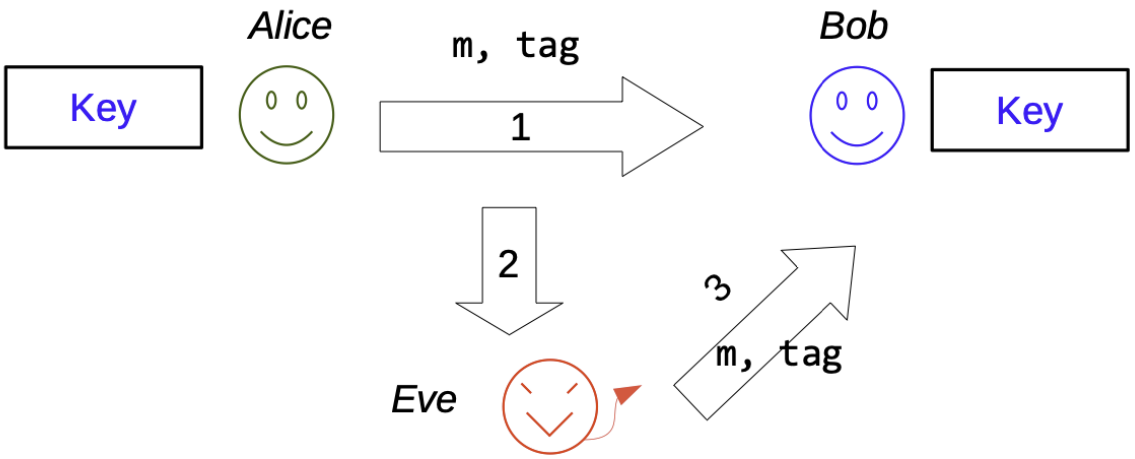
- md5 --> insicuro e deprecato --> non è più una funzione crittografica perchè sono state trovate collisioni
- **sha** --> standard per le funzioni hash (la famiglia sha1 è deprecata). La dimensione del digest è dipendente dalla funzione che si utilizza (sha256 o sha512)

Funzioni MAC

Message Authentication Code --> generalmente chiamate "funzioni hash con chiave". la differenza concettuale con le funzioni hash è che le funzioni MAC accettano una chiave. Con questo approccio si riesce a garantire sia **Integirtà** che **Autenticità**:

- Sono funzioni che mappano un messaggio di lunghezza arbitraria all'interno di un tag di lunghezza fissa
- Funzioni sicure alla Lenght-extension --> comunque alcune sono basate su funzione HASH come le HMAC

Replay Attack



In questo contesto il destinatario accetta più messaggi uguali non inviati dal mittente, ma duplicati da `Eve`. Questo dimostra che il protocollo MAC garantisce l'autenticità del singolo messaggio, non del flusso di comunicazione

Alcune difese

- anche nei protocolli sicuri si utilizza un IDENTIFICATIVO, in modo da riuscire a rilevare i replay attack.
- Se ci si appoggia su un protocollo siuro (TCP) si possono sfruttare elementi come il sequence number.

AEAD

Cifratura autenticata con dei messaggi associati di cui si garantisce solo l'autenticità

Problemi

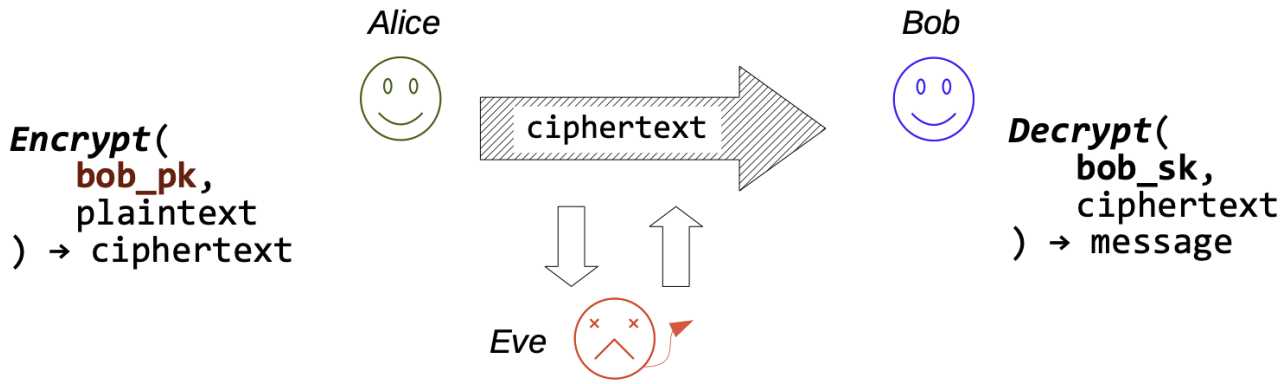
Il problema della crittografia simmetrica è quello di scambiarsi una chiave segreta condivisa, nasce così la **crittografia asimmetrica**

Crittografia Asimmetrica

A differenza della crittografia simmetrica qui esiste una coppia di chiavi: **secret-key** e una **public-key** e ogni parte della connessione deve avere una coppia di chiavi. In base allo scopo della comunicazione abbiamo diversi utilizzi.

Lo schema è il seguente:

- `encrypt(public-key, message) → ciphertext` --> la chiave che si usa per cifrare è la chiave pubblica del destinatario
- `decrypt(secret-key, ciphertext) → message` --> la chiave privata è la propria, ma ovviamente il messaggio deve essere stato cifrato con la propria chiave pubblica



Così facendo garantiamo riservatezza, per garantire l'autenticità invece devo avere una **firma digitale**:

- `sign(secure-key, data) → signature`
- `verify(public-key, data, signature)`

Il procedimento quindi risulta che:

- si cifra un messaggio con una chiave asimmetrica
- si crea una signature sul messaggio cifrato
- si inviano entrabe al destinatario, che verificherà tramite sua chiave privata il messaggio cifrato e l'autenticità del messaggio tramite chiave pubblica del mittente

KEM

Lo scenario più comune di utilizzo della crittografia asimmetrica è quello di scambio delle chiavi simmetriche. Per quanto più sicuro l'utilizzo di chiavi asimmetriche è molto più "pesante" rispetto all'utilizzo di chiavi simmetriche:

- **KEM** --> cifro la chiave simmetrica in maniera asimmetrica e invio. Una volta decifrata la chiave uso quello per il resto della comunicazione

Men in the middle

- Quando una entità malevola si mette in mezzo e manipola il canale o la comunicazione
- In questo scenario se non si firma si viola la confidenzialità. basta la firma per riconoscere uno schema del genere

Trusted third party

Obiettivo:

- Autenticità della chiave pubblica

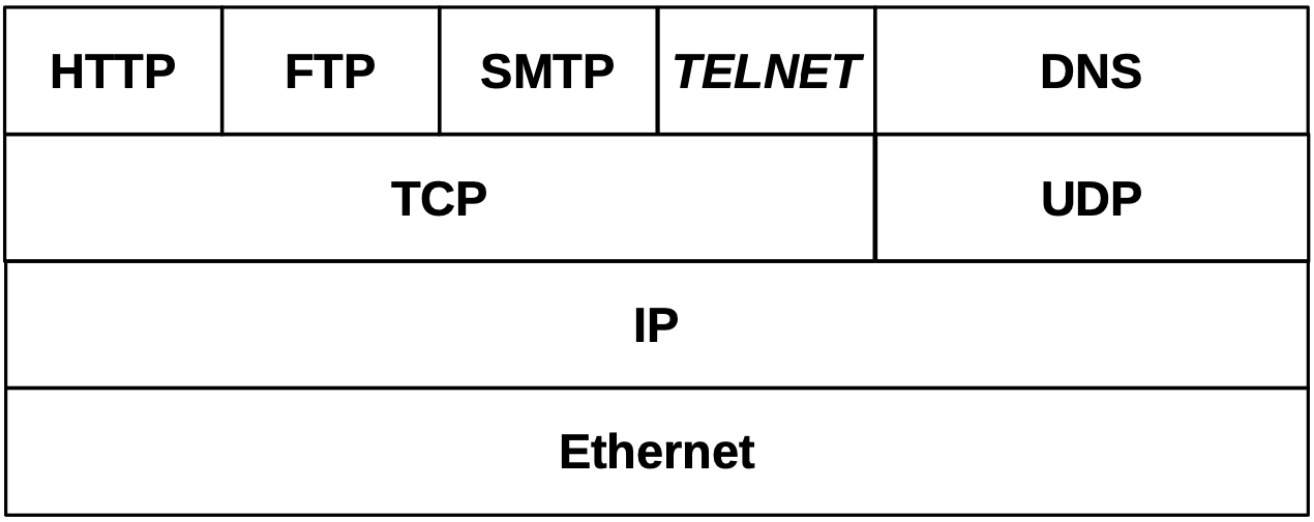
Utilizziamo una terza parte chiamata **Certification Authority** per certificare l'autenticità della chiave pubblica che stiamo distribuendo. La "fiducia" è data dal fatto che entrambi i capi della connessione conoscano la chiave pubblica della CA. il CA fima le chiavi pubbliche della comunicazione in modo da garantire Autenticità.

Il certificato include dei metadati utili al protocollo:

- associa sia informazioni crittografiche: chiave pubblica e firma della CA
- a informazioni del tipo: identità, validità, contesto di utilio

Nella realtà c'è una CA ROOT che delega il lavoro a CA intermedie. Così facendo si creano delle catene di certificati: ovvero un insieme di certificati in cui noi verifichiamo l'autenticità grazie. La parte di CA sono fatte OFFLINE --> ovvero Prima di instaurare la connessione

Protocolli sicuri



TLS

Su https --> c'è lo stack normale TCP, aggiungendo un layer sicuro con un protocollo TLS:

- TCP crea una connessione
- TLS aggiunge sicurezza

> Se c'è un protocollo con la `s` finale vuol dire che è stato aggiunto un layer di sicurezza

IPSec

Protocollo che estende il layer IP --> layer che agisce sotto il livello 4:

- nel protocollo sicuro in generale gli header dei livelli sottostanti rimangono uguali --> garantiamo confidenzialità solo dopo il livello trasporto

```
| Header H2H | Header IP | Header TCP | Header TLS | PAYLOAD |
```

Esempio in TLS i dati cifrati partono dall'header TLS e non prima, più agisco in alto meno riesco a nascondere.

IPSEC implementa confidenzialità a partire dall'header del TCP, questo sicuramente rende più difficile la gestione del pacchetto

MACSec

Analogamente è un protocollo che aggiunge un layer di autenticità e opzionalmente di confidenzialità a livello H2N