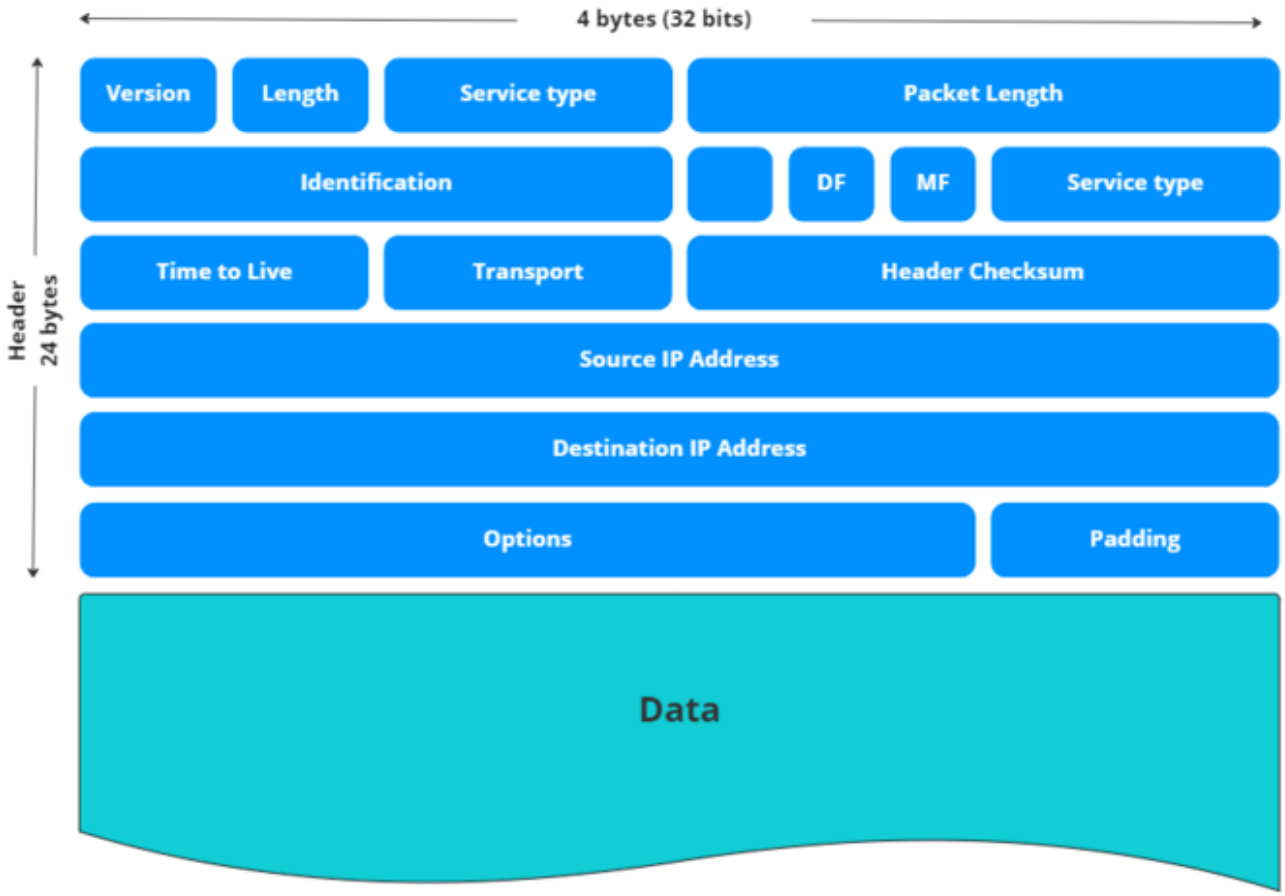
Applicazione della crittografia alla cybersecurity

Le principali **applicazioni di crittografia sono legate ai protocolli di comunicazione**. Ci sono altre applicazioni come crypto file systems che consente di cifrare i dati sul hard.

Network Security: IPSEC

Il protocollo principale che usa la **crittografia a livello di rete è IPSEC**, originariamente è stato un sotto modulo sviluppato per IPv6 poi è stato adottato anche dal protocollo IPv4.



L'**obiettivo** di IPSEC è quello di **cifrare e autenticare** il traffico a livello IP, **è una risposta agli attacchi di intercettazione e agli attacchi di spoofing**.

> Ultimo RFC 4301-4307 (december 2005)

Gli algoritmi usati all'interno dell'IPSEC sono (RFC 4835)

- AES CBC 128bit
- Triple DES CBC 168bit
- Message Authentication / integrity
  - SHA1/SHA2 HMAC
  - MD5 (deprecato)

Servizi offerti da IPSEC

- Data origin authentication: autenticazione del mittente, previene ip spoofing,
- Confidenzialità,
- Integrità,
  - Sul singolo pacchetto,
  - Partial sequence integrity, previene packet replay (replay attack).
- Possibile nascondere l'indirizzo IP del mittente limited traffic flow confidentiality

La cosa **importante è che questi servizi sono trasparenti alle applicazioni** (a livelli sopra transport layer), cioè che tutti questi servizi vengono forniti a livello di sistema operativo, il programmatore che scrive applicazioni in una macchina che contiene ipsec non deve preoccuparsi del fatto che i suoi vengano intercettati sulla rete e dell'ip spoofing. Ovviamente tutto questo vale fino che il pacchetto arrivi alla macchina di destinazione, una volta decifrato il system admin può vedere il contenuto.

IPSEC Protocols

IPSEC viene sviluppato con due tipi di protocolli:

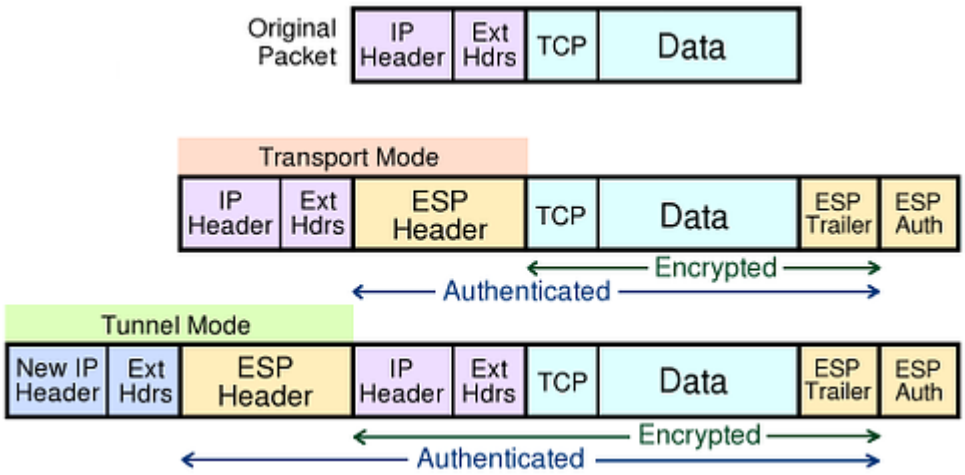
- **Authentication Header** - AH: fornisce servizi di **integrità**, fa un **controllo di integrità su tutto il pacchetto**,
- **Encapsulated Security Payload** - ESP: fornisce servizi di **confidenzialità**. Può fare sia **encryption** che **authentication**, a differenza di AH, ESP fa il **controllo di integrità sul payload ma non sul header**.

In realtà, con l'utilizzo di IPSEC si sono resi conto che bastava ESP. AH è stato fatto downgraded da MUST a MAY, quindi, non è più obbligatorio.

IPSec aggiunge, un ipsec header, cioè che modifica il formato del pacchetto, quindi, può essere capito solo da un'altra macchina che ha ipsec.

Tunnel Mode / Transport Mode

IPSEC può operare in due modalità:



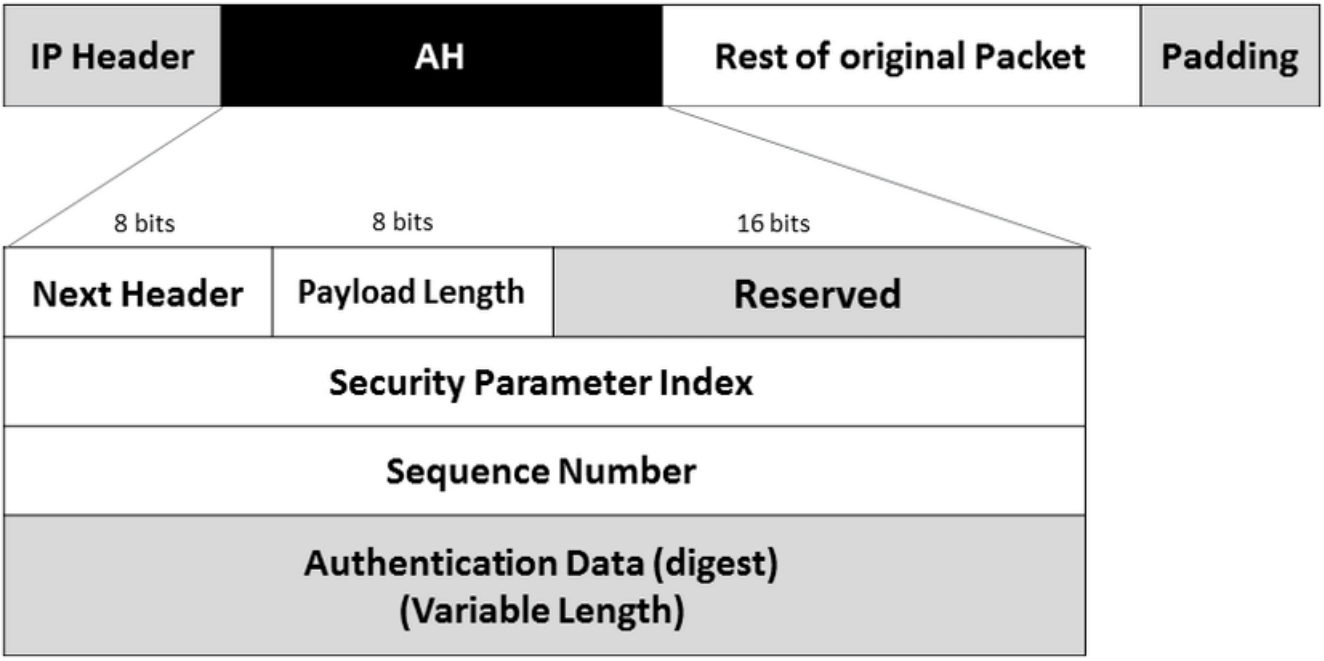
- **Tunnel mode**: il pacchetto ipsec viene **wrappato in un pacchetto IP**, questo viene fatto per reggere la transizione da IPv4 a IPv6 e per far viaggiare il pacchetto su una rete che non supporta IPSEC. Es. far comunicare le macchine con ipsec con quelli **senza ipsec**. Tunnel mode è usato con VPN.

- **Transport mode**: usato per inviare pacchetti tra due **macchine con ipsec**.

Authentication Header - AH

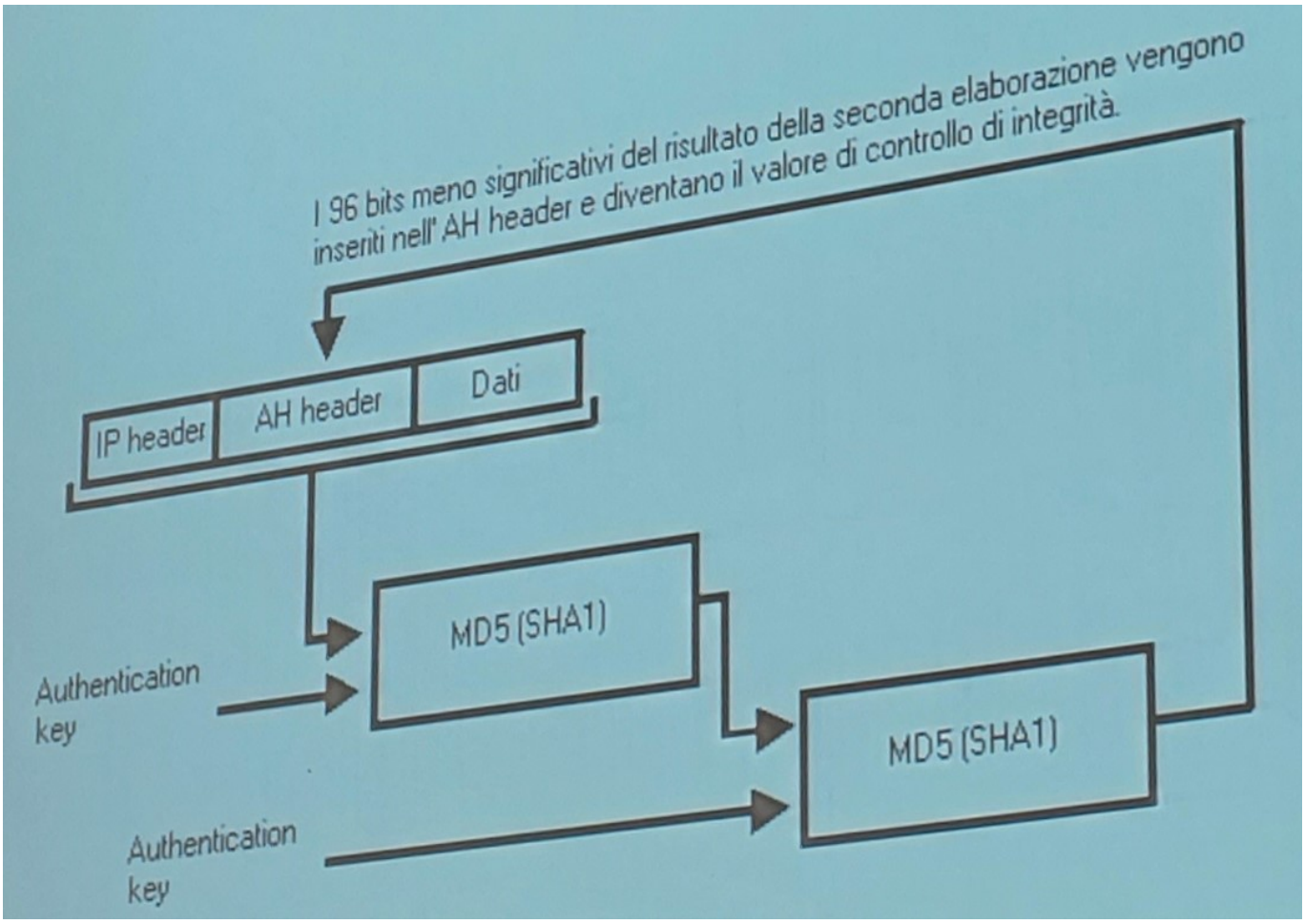
L'header di AH contiene : Security Parameter Index **SPI** ed un Integrity Check Value **ICV** (Authentication Data).

Il payload è dato da: se **tunnel** mode da **IP + Data** e se **transport** mode da **TCP/UDP + Data**.



Come funziona l'integrity check?

Si prende un pacchetto IP, si prende AH header, **si prendono i dati ed attraverso una chiave di integrità si fa doppio hash** (SHA1) e il risultato viene inserito nell'AH.

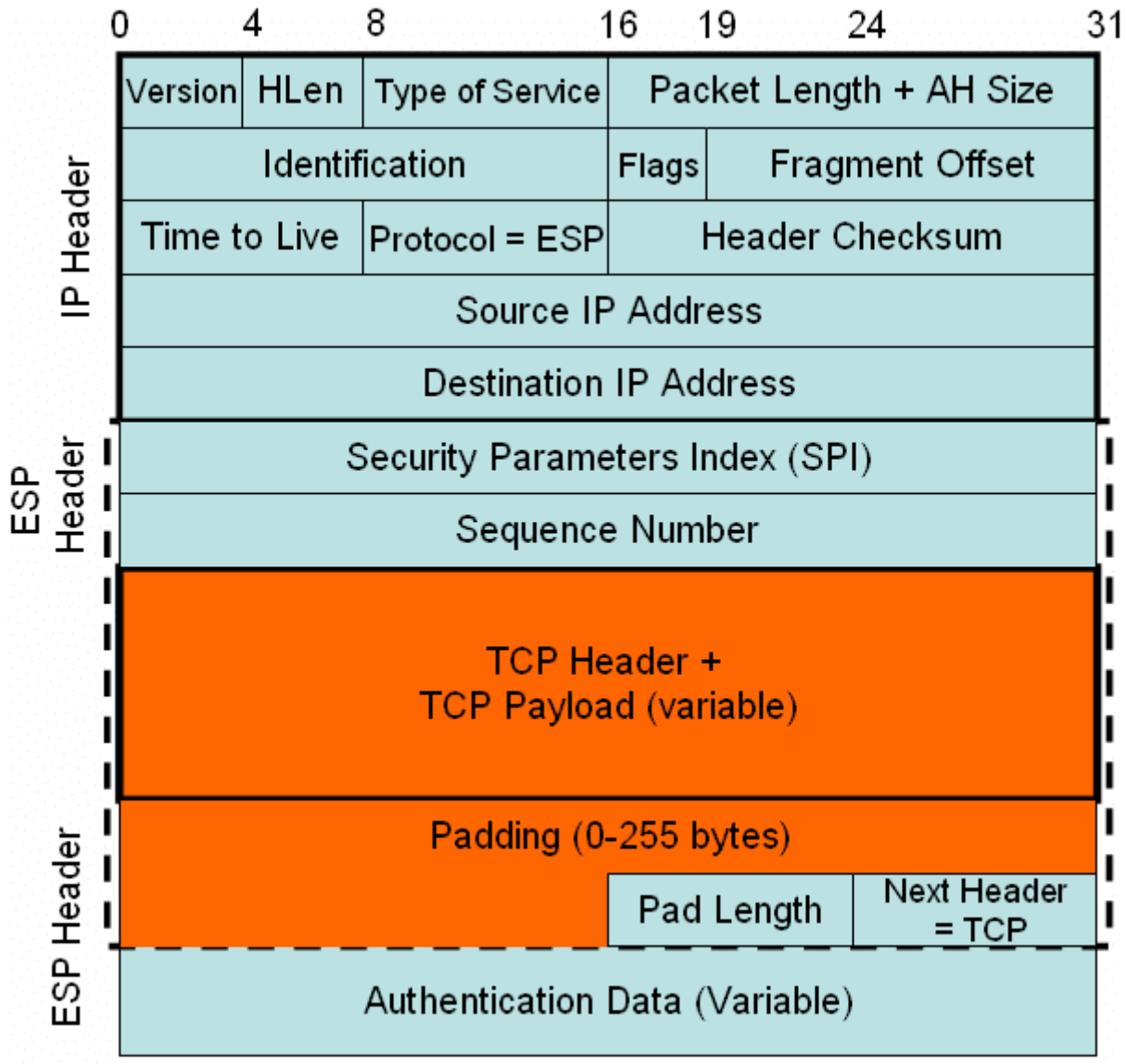


Integrity check value viene inizializzato a zero per calcolo del hash e poi aggiornato con il valore calcolato (HMAC).

Chi riceve il pacchetto, rifa i calcoli per controllare l'integrità.

Encapsulated Security Payload - ESP

ESP garantisce confidenzialità e integrità del payload.



Il router capisce che stiamo lavorando dal numero di protocollo. IPSEC protocollo 50 per ESP e 51 per AH.

ESP può essere usato da solo come cifratura oppure cifratura + autenticazione.

- **Solo cifratura** : Cifra il **payload e IPv6 extension header** ed in tunnel mode cifra tutto il pacchetto IP,
- **cifratura + autenticazione**: aggiunge il controllo di **integrità al payload**.

AH autentica IP payload e alcune parti del header in transport mode ed in tunnel mode autentica tutto il pacchetto.

	transport mod	**tunnel mod**
AH	Authenticates IP payload and selected portions of IP header and IPv6 extension headers	Authenticates entire inner IP packet plus selected portions of outer IP header
ESP	Encrypts IP payload and any IPv6 extension header	Encrypts inner IP packet

	transport mod	tunnel mod
ESP with authentication	Encrypts IP payload and any IPv6 extension header. Authenticates IP payload but no IP header	Encrypts inner IP packet. Authenticates inner IP packet but no outer IP header

Come viene implementato?

Idea: bisogna lavorare a livello di sistema modificando il codice di protocollo IP, cambiando il comportamento in base al tipo di pacchetto se deve fare trasformazioni crittografiche oppure no.

In pratica: si parte con una richiesta **all'utente di definire un Security Policy Database**.

Security policy database

Dice cosa fare al traffico. Es. tutti il traffico dalla mia macchina ad un server particolare me lo deve cifrare.

Le SP specificano: che **tipo di trasformazioni fare sul pacchetto**, se andiamo in tunnel mode oppure in transport, se fare una trasformazione crittografica sul quel traffico, trasformazione che vogliamo effettuare sul pacchetto, ecc.

Opzioni:

- `--sp-source` = source address
- `--sp-destination` = destination address
- `--source-port`
- `--destination-port`
- `--upper-layer-protocol` = TCP | UDP | ICMP6 | ANY
- `--flow-direction` = in | out
- `--action` = none | discard | ipsec
- `--sp-mode` = tunnel | transport
- `--sa-name` = security association name
- `--sp-name` = security policy name

Va definito per ogni tipologia di traffico, cioè per ogni copia di ip source e destination.

Questi informazioni non sono sufficienti. Che trasformazione crittografica devo fare? Controllo d'integrità e/o encryption, quale algoritmo usare? es. SHA1, SHA2.

Security Association database

Dice come implementare le policy, cioè, dice per ogni coppia di host che cosa bisogna fare.

Perché sono divisi in due? perché in caso di vulnerabilità di un algoritmo crittografico basta cambiare il codice, altrimenti bisogna modificare anche la policy.

Opzioni:

- `--sa-source` = source address
- `--sa-destination` = destination address
- `--sa-mode` = tunnel | transport

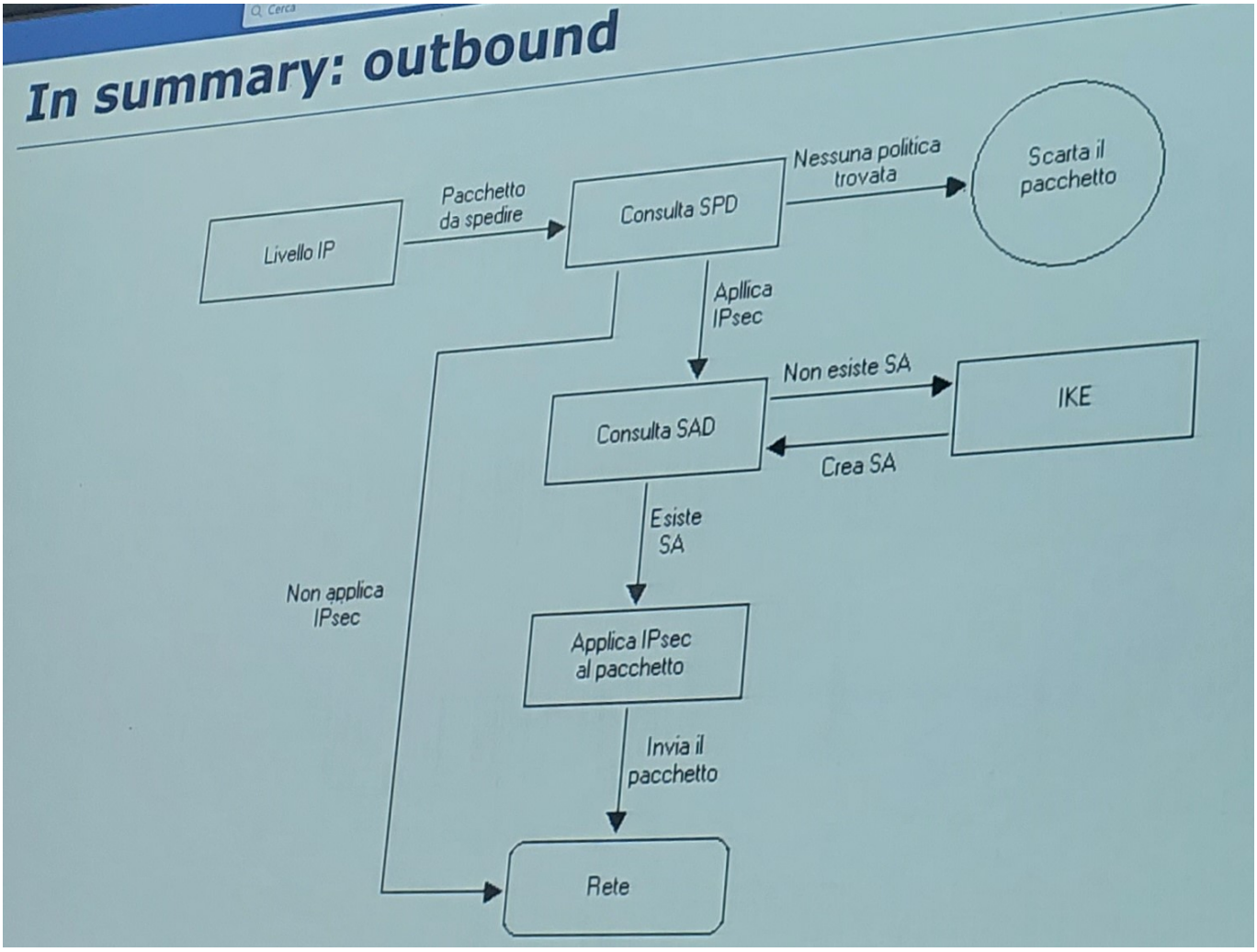
- `--sa-spi` = security parameter index
- `--encryption-algorithm` = 3des-cbc | aes128-cbc | null
- `--encryption-key`
- `--integrity-algorithm` = hmac-sha1 | hmac-sha2-256
- `--integrity-key`
- `--sa-name` = security association name

Internet Key Exchange Protocol - IKE

All'inizio la configurazione era manuale, quindi, bisognava mettersi d'accordo con le persone con si comunicava. **Introducendo il protocollo IKE è stato automatizzato il processo di configurazione**. è un protocollo che viene attivato inconsapevolmente all'utente.

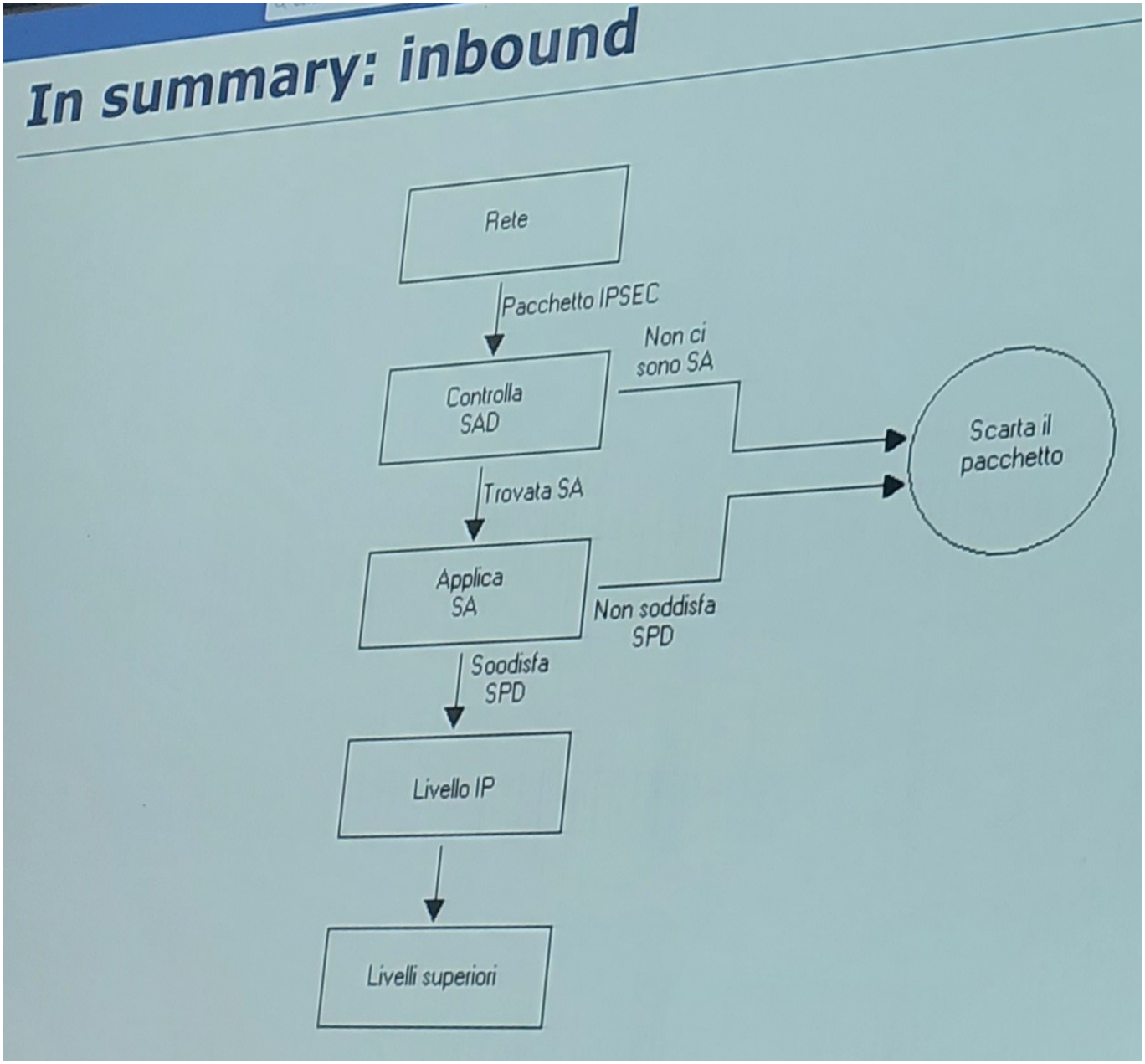
IKE Outbound

A livello IP arriva il pacchetto da spedire **con IP Source/Destination con questi campi si accede alla ricerca del Security Policy Database** (source, destination, port), se devo applicare ipsec consulto il corrispondente record sul Security Association Database, se non è definito IKE lo definisce contrattando con il destinatario. Nel momento in cui ho SA applico al pacchetto le trasformazioni richieste e invio nella rete.



IKE Inbound

Il pacchetto ipsec contiene il parametro Security Parameter Index **SPI** che **viene usato in ricezione per accedere al Security Association Database**, quando ricevo il pacchetto non so fare la trasformazione (decifrarlo), una volta trovato vado vedere se **SA combacia con Security Policy e applico trasformazioni, altrimenti scarto il pacchetto**.



Benefici di IPsec

- **fornisce un buon livello di sicurezza a tutti livelli superiori a TCP**,
- **trasparente a TCP**, a tutte le applicazioni, al programmatore e all'utente finale,
- **facilmente estendibile a nuovi protocolli crittografici**

Svantaggi IPsec - Drawbacks

- **Rallentamento del traffico perché la cifratura costa** (anche se simmetrica é veloce) (ike usa Diffie-Hellman),
- **messaggi broadcast non si possono fare** perché serve un SA con ogni dispositivo della rete.

Transport Level Security - TLS / SSL

IPsec nasce dopo TCP security, il primo protocollo ad affrontare il problema di sicurezza sulla rete è stato SSL poi diventato TLS. è un protocollo sviluppato da Netscape e standardizzato con RFC 2246. Ha come obiettivo quello di stabilire un canale sicuro tra due host a livello di trasporto.

Se IPsec è a livello IP, allora non dovrebbe leggere la porta che è all'interno del header TCP. In realtà, per non sovraccaricare IPsec è stato usato il concetto di porta anche in IPsec, es. posso cifrare solo il traffico HTTP.

TLS nasce a livello TCP, quindi cifra la connessione da una porta a porta. è lo standard di riferimento per le connessioni HTTP.

La version attuale è TLS è 1.3 RFC 8446 (AUG 2018)

Servizi del protocollo TLS

- Server authentication
- Client authentication (opzionale)
- Data confidentiality, contro intercettazione
- Data integrity, contro le modifiche
- Non fornisce non-ripudio
- Generation / Distribution of session keys
- Security parameter negotiation
- Compressione e decompressione

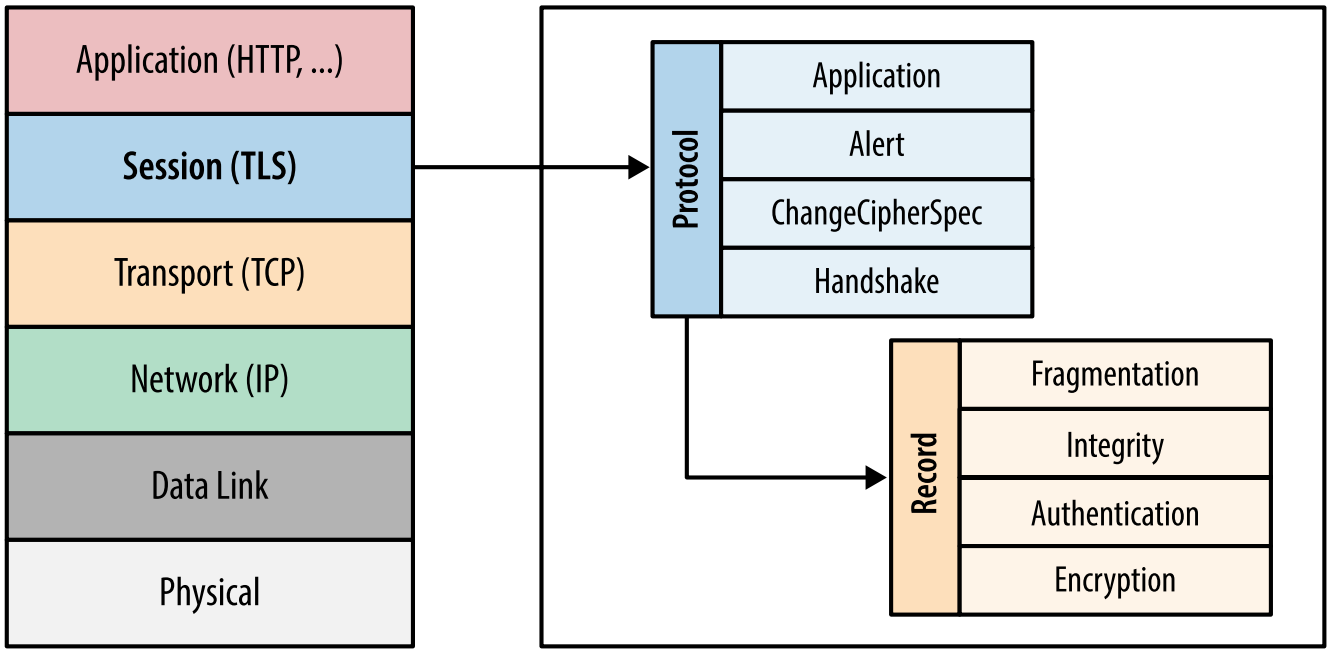
Come funziona? SSL / TLS standard

Per l'autenticazione usa certificati x.509. Algoritmi usati per key exchange sono `RSA`, `DH`, `DSS`, `Fortezza`.

Encryption algorithms: RC4 (40-128), DES (40-128), 3DES, AES

Algoritmi Hashing usa: MD5 (deprecato), SHA1, SHA2

Layer TLS viene posto sopra il layer TCP.



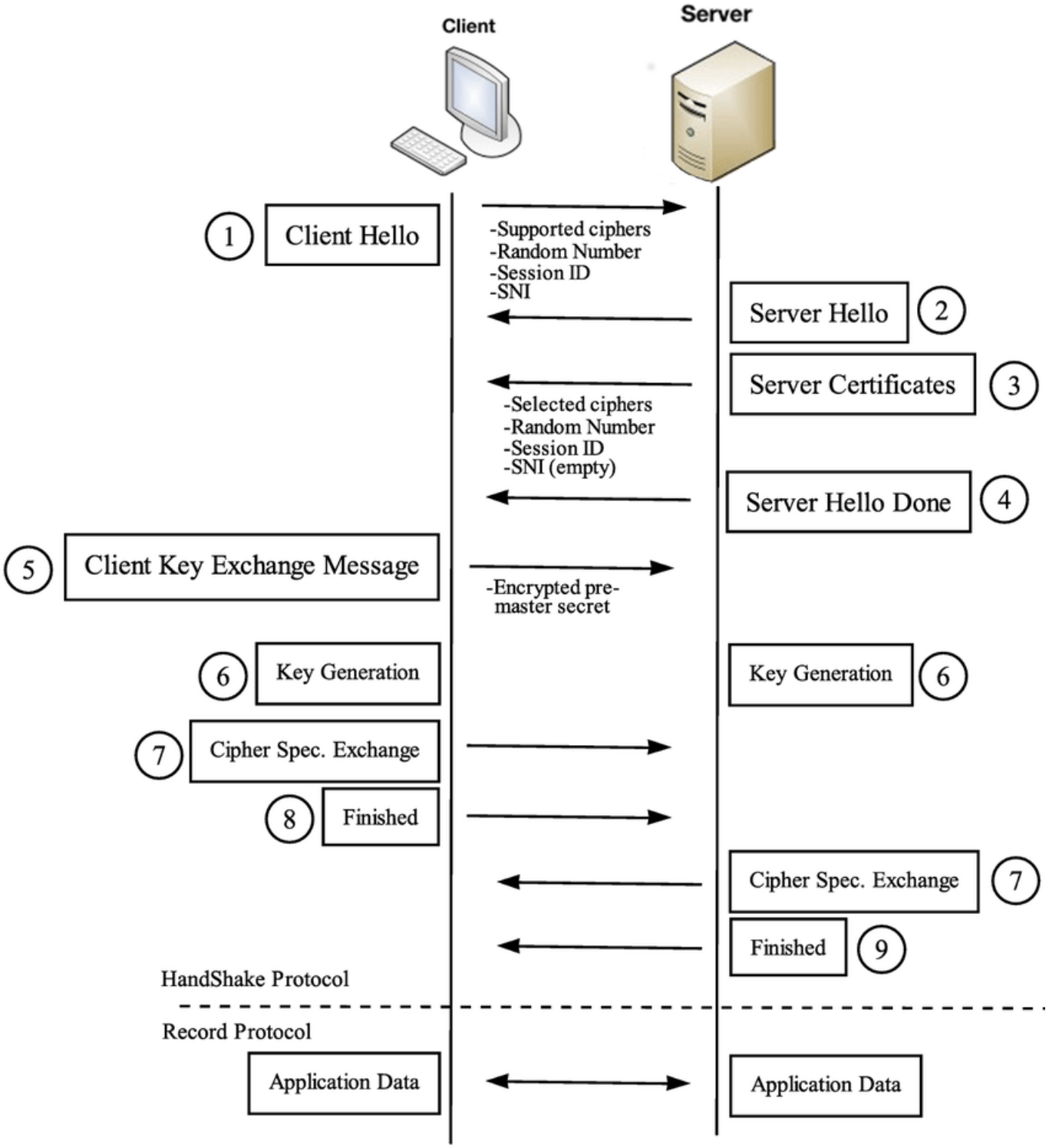
Sotto protocolli del TLS - Handshake protocol, Record protocol

- SSL **Handshake protocol**, è usato per stabilire le chiavi tra client e server. nella fase di handshake i due si mettono d'accordo sulle trasformazioni da fare es. algoritmi crittografici da usare e lo scambio di chiavi.
- SSL Change Cipher Spec protocol
- SSL Alert protocol
- SSL **Record protocol**, opera sulle trasformazioni contrattate nella fase di handshake.

Handshake

Come funziona handshake protocol?

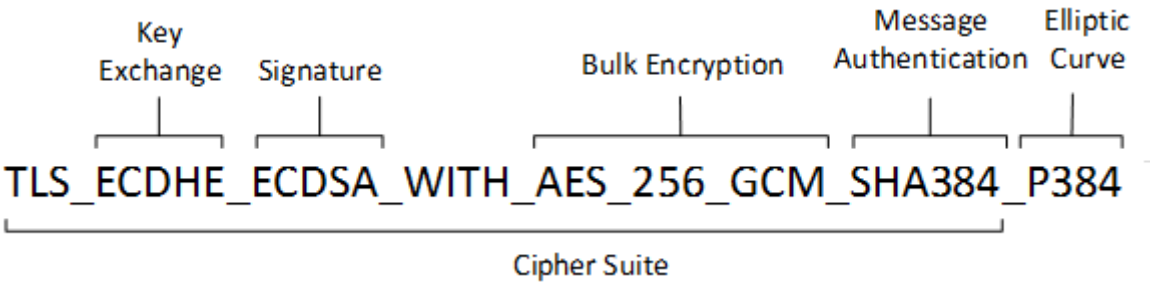
1. Client **invia un** `numero casuale` **e un oggetto chiamato** `cipher_suite` ed altre informazioni (come time) al server.
2. Il server risponde a sua volta con un altro **numero random, una cipher_suite** (confermando o meno cipher suite del client in questo caso propone un suite più adatto al server), session id e la chiave pubblica del server.
3. Client prende la chiave pubblica del server e calcola la `premaster_secret`.



Cipher Suite

Cipher Suite sono due byte che rappresentano le modalità con cui client e server vogliono comunicare.

```
Cipher Suite: TLS_AES_128_GCM_SHA256 (0x1301)
Cipher Suite: TLS_AES_256_GCM_SHA384 (0x1302)
Cipher Suite: TLS_CHACHA20_POLY1305_SHA256 (0x1303)
Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b)
Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xc02c)
Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)
Cipher Suite: TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca9)
Cipher Suite: TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca8)
Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)
Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)
Cipher Suite: TLS_RSA_WITH_AES_128_GCM_SHA256 (0x009c)
Cipher Suite: TLS_RSA_WITH_AES_256_GCM_SHA384 (0x009d)
Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0x002f)
Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA (0x0035)
Cipher Suite: TLS_RSA_WITH_3DES_EDE_CBC_SHA (0x000a)
```



Come si calcola il master secret

Una volta scelto algoritmi calcolano il master secret.

Il premaster secret: Il client genera un numero casuale ed invia al server, quindi, il numero casuale è condiviso tra i due host.

```
master_secret = SHA1(premaster_secret + SHA1('A' + premaster_secret +
ClientHello.random + ServerHello.random)) ||
SHA1(premaster_secret + SHA1('BB' + premaster_secret + ClientHello.random +
ServerHello.random)) ||
SHA1(premaster_secret + SHA1('CCC' + premaster_secret + ClientHello.random +
ServerHello.random))
```

Da cui si ottiene un key block da 6 chiavi, 4 chiavi e 2 vettori di inizializzazione:

- client write mac
- server write mac
- client write
- server write
- client write IV (initialization vector per cbc)
- Server write IV

Client cifra con una chiave diversa da server. Così come usano due chiavi diverse per MAC (message authentication code). Infine, il vettore di inizializzazione che serve per AES-CBC cipher block chaining.

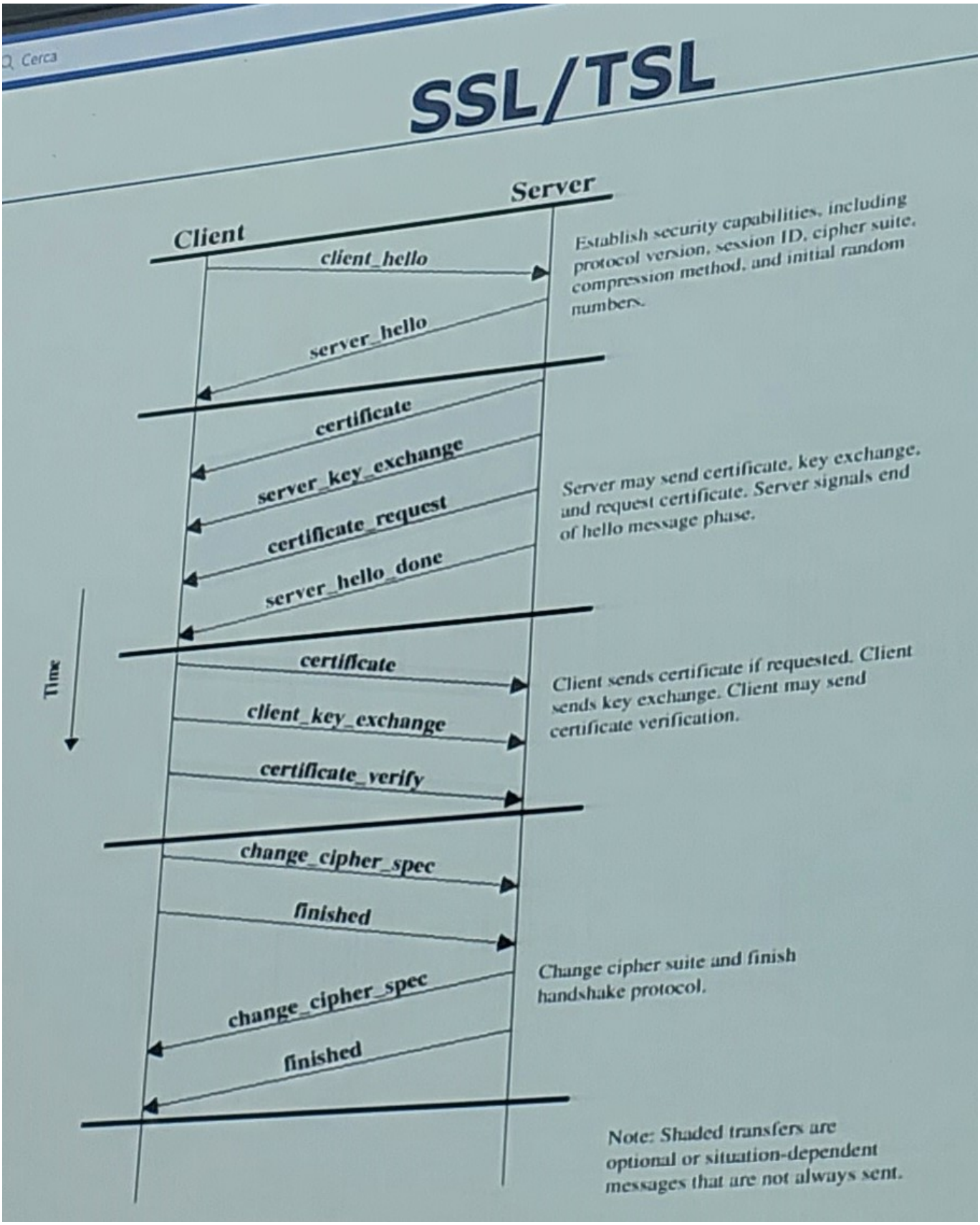
Ci può essere un problema di MITM, nel senso che quando il server manda la chiave pubblica ci può essere un MITM che intercetta e ne manda un altra chiave al client.

Per risolvere MITM, si usano certificati, quindi, server manda oltre alla chiave pubblica anche il certificato firmato da CA.

Autenticazione attraverso il certificato è obbligatoria ma anche il server può opzionalmente richiedere al client il certificato.

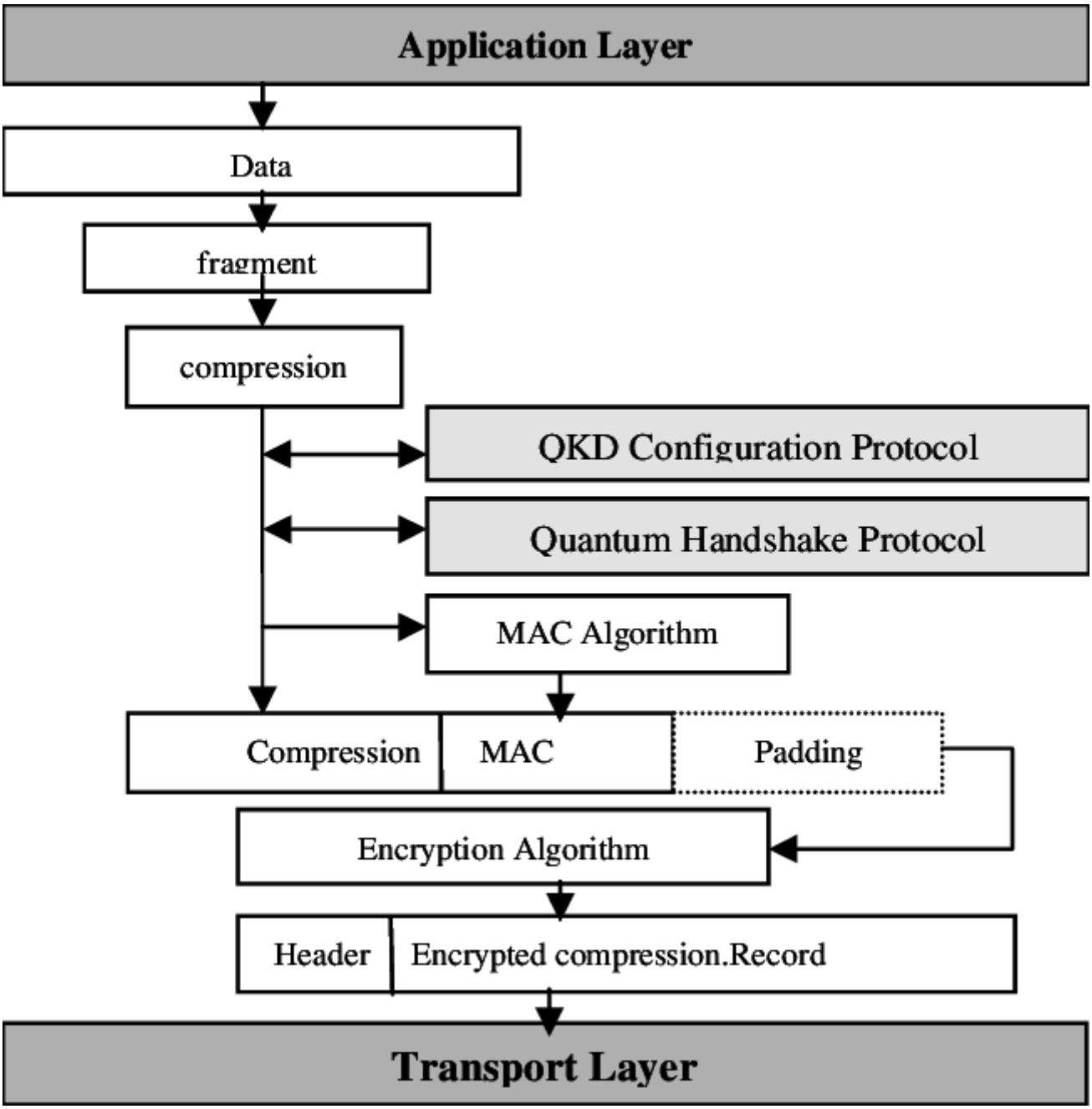
Normalmente, per una questione di praticità non si fa mutua autenticazione ma solo autenticazione del server. (il certificato deve essere rinnovato, comprato, pubblicato sulla gazzetta ufficiale, ecc).

Il protocollo Handshake è tutto automatizzato, infatti, IPsec con IKE incorpora meccanismi simili del handshake di TLS.



Record protocol

Una volta definiti le chiavi sono state definite con protocollo handshake, **il record protocol prende il pacchetto TCP, lo frammenta, fa eventualmente una compressione, calcola Message Authentication Code MAC, fa la cifratura, aggiunge header TLS ed invia al layer successivo**.



HTTPS è la versione di HTTP che utilizza TLS.

Vulnerabilità di TLS

Rispetto IPsec, il principale criticità di TLS è il fatto che **non è più trasparente rispetto ad applicazione**. Se si vuole fare una applicazione che fa una cifratura del traffico bisogna modificarlo in modo da usare TLS invece di TCP, mentre con IPsec, se un'applicazione viene montato su un sistema con IPsec da quel momento è siamo sicuri che tutto il traffico generato dall'applicazione sarà cifrato. Invece, se la macchina non ha IPsec, la cifratura bisogna metterlo a livello applicazione. Quindi, **TLS non è trasparente ad applicazione, per cui tutte le applicazioni che vogliono usare cifratura, controllo di integrità a livello di trasporto devono essere modificati**.

- Beast 2011: è un exploit del browser che utilizza una vulnerabilità del CBC per estrarre plaintext da una sessione cifrato.
- Heartbleed 2014: rubava private keys da server che usava una versione vulnerabile di OpenSSL.

End-to-End Encryption - EE2E

Come funziona Gmail? Gmail (email in chiaro) -> TLS (cifra il email) -> TCP -> IP.

L'email in chiaro passa a TLS che lo cifra passa a TCP ed ad IP.

Amministratori di server di Gmail possono leggere l'email (viene letto tra TLS e livello applicazione Gmail).

In direzione inversa, da IP passa a TCP che a sua volta passa a TLS che lo decifra e passa l'email in chiaro all'applicazione.

Per non essere letto bisogna usare una **end-to-end encryption, viene decifrato a livello dell'applicazione, quindi, solo l'utente può leggere il messaggio in chiaro**.

End-to-End viene usato generalmente dalle applicazioni messaggistiche come Whatsapp, Telegram e Signal. Ma GPG permette di usarlo anche per email.

Telegram usa il protocollo **MTProto**, Whatsapp e Signal usa il protocollo **Signal**.

Il **problema dell'algoritmo di Diffie-Hellman è che richiede che i due utenti si interagiscano per generare l'informazione segreta comune tra i due** ma non sempre la persona che riceve un messaggio è raggiungibile. Quindi, servirebbe un Diffie-Hellman non interattivo.

Protocollo MTProto

è un **insieme di protocolli crittografici che permette di avere un cifratura end-to-end**, progettato per Telegram. Presuppone che tutto quello che sta sotto il livello di applicazione è non-sicuro.

Prevede due modalità: cloud chats e secret chats.

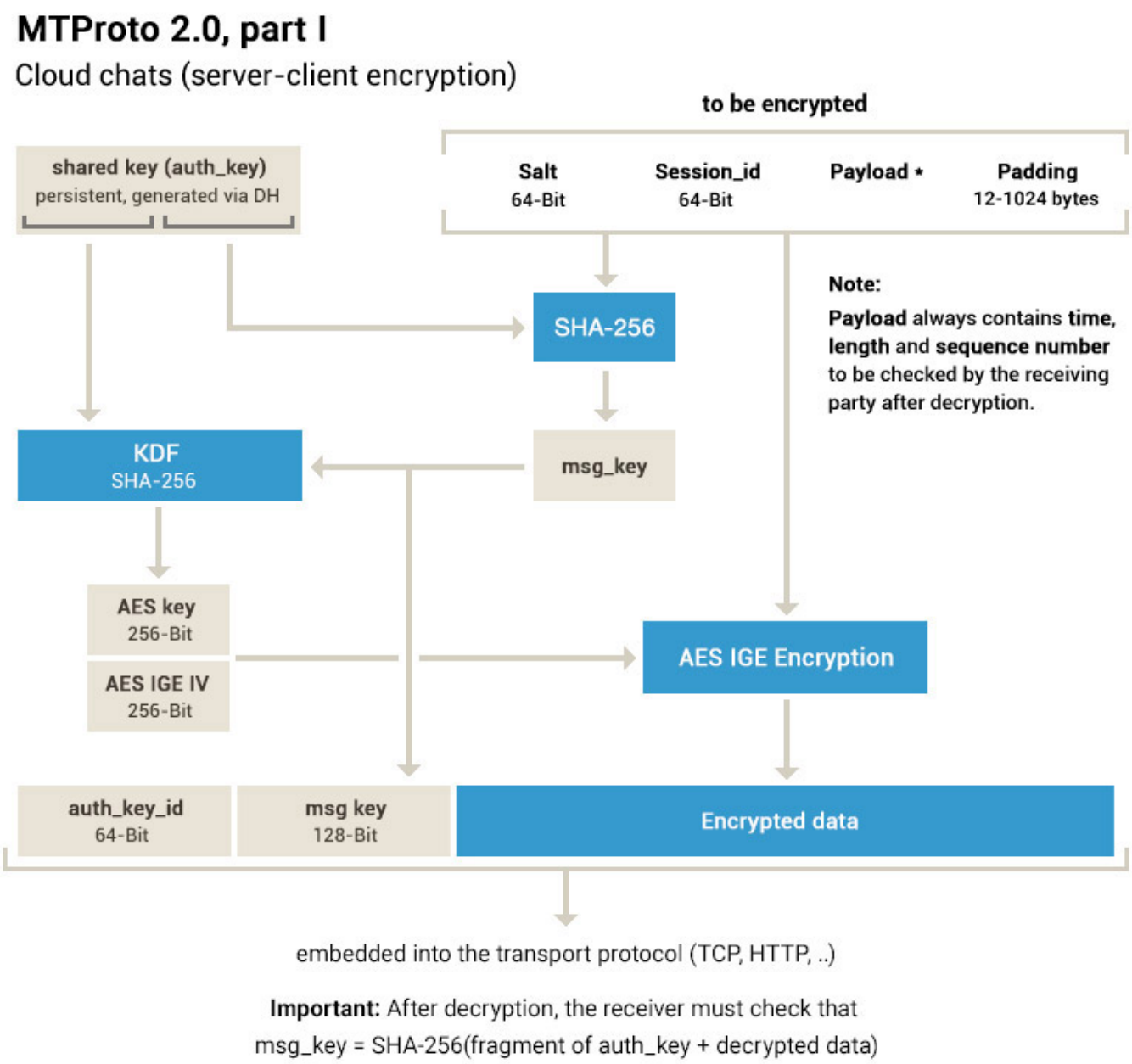
In caso di **cloud chats, i messaggi sono memorizzati in chiaro sul server invece secret chats sono memorizzati cifrati**.

Gli **utenti non comunicano direttamente tra di loro ma attraverso un server**. Il messaggio viene memorizzato sul server finché non viene scaricato da un device.

Cloud chats - MTProto 2.0

Alice A_s e Bob B_s attraverso Diffie-Hellman stabiliscono due chiavi AES segrete con il server.

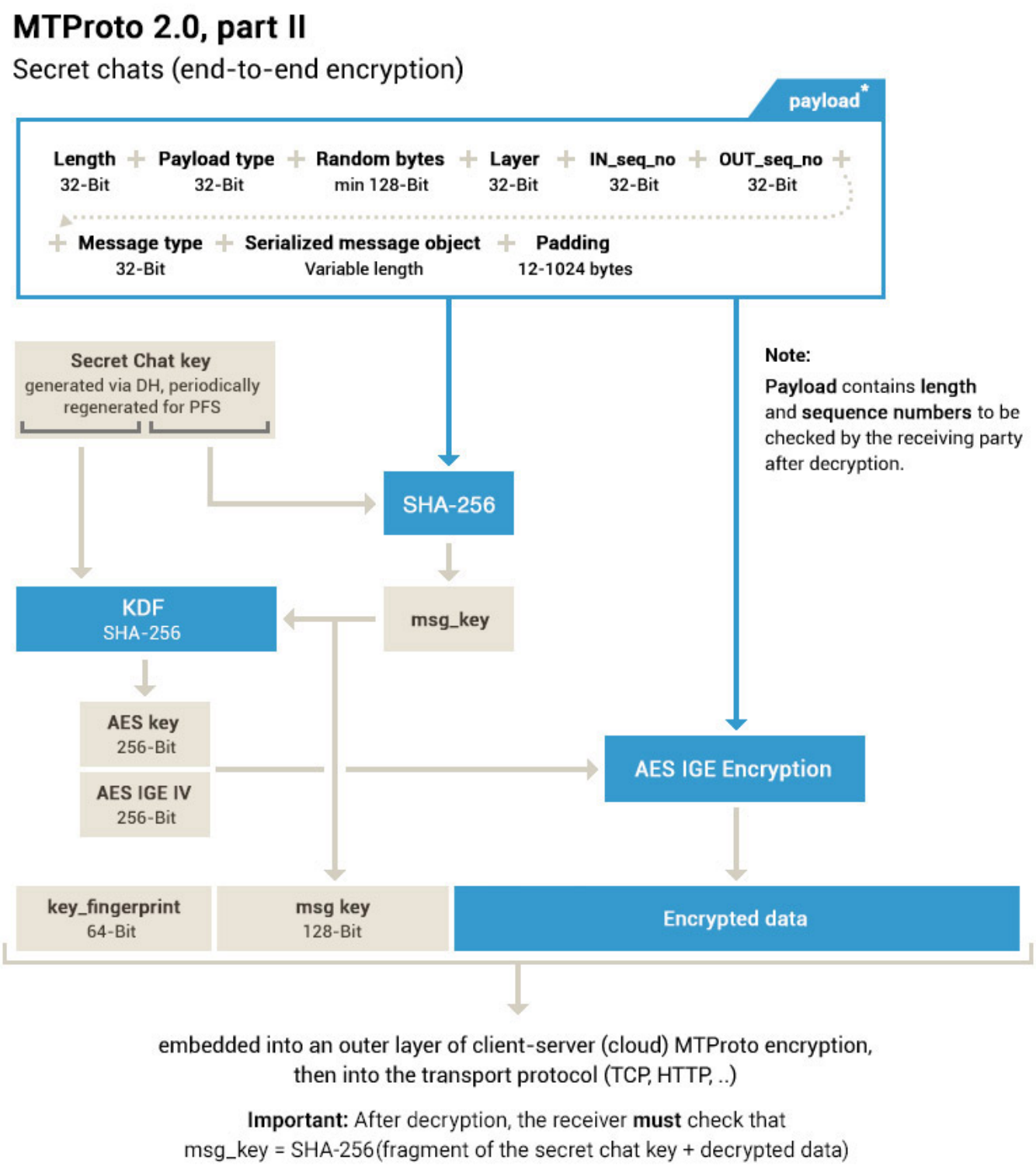
Quando Alice deve comunicare con Bob, dice al server che voglio comunicare con Bob al server e gli manda il messaggio cifrato con la chiave segreta di Alice A_s, il server lo decifra e lo cifra con la chiave di Bob B_s.



Secret chats

Nelle secret chat **DH viene fatto direttamente tra A e B** senza il server di mezzo come nel caso di cloud chats ma i messaggi vengono inviati attraverso un server comunque.

Problema è che il server potrebbe fare MITM. Si assume che il serve sia fidato.



Signal protocol

Protocolli crittografici utilizzati da Signal:

- Elliptic Curve Diffie-Hellman ECDH `curve25519`.
- Extended Triple DH Agreement X3DH,
- AES,
- HMAC SHA256,
- Double Ratchet Algorithm.

Il protocollo può essere suddiviso in 4 fasi:

- Registrazione
- Session Set-up
- Symmetric-ratchet communication
- Asymmetric-ratchet communication

Registration Fase

Bob pubblica una serie di chiavi pubbliche sul server durante la fase registrazione (Identity Key IKb, Signed prekey SPKb).

Session set-up Fase

Nel momento in cui Alice vuole comunicare con Bob, accede alle informazioni pubbliche memorizzati di Bob sul server, quindi, recupera informazioni crittografici necessari, calcola tre chiavi DH e una chiave SK tramite Key derivation function KDF.

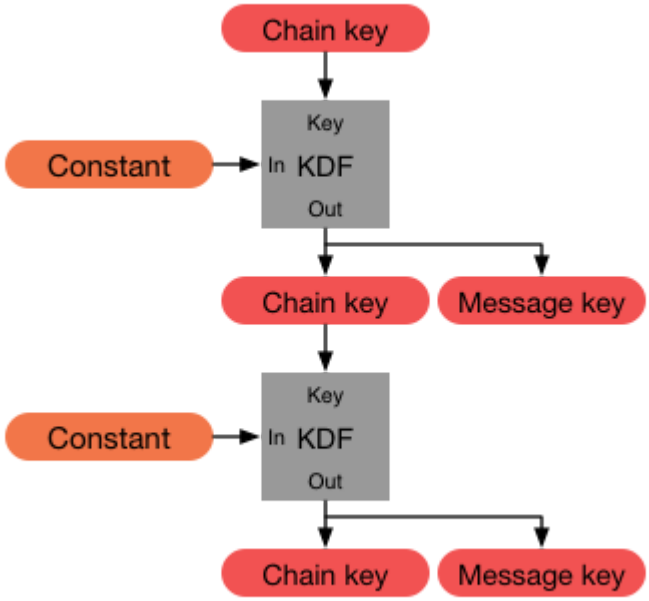
```
DH1 = DH(IKa; SPKb)
DH2 = DH(EKa; IKb)
DH3 = DH(EKa; SPKb)

SK = KDF(DH1 || DH2 || DH3)
```

La chiave SK viene usato per cifrare il messaggio da inviare. Quindi, **le chiavi sono generati tramite un misto di informazioni di A e un misto di informazioni di B**.

B per **decifrare accede alle informazioni pubbliche di A** e rifa i conti.

Key Ratchet

Un algoritmo che prende una chiave e ne genera delle tante altre. Ad ogni messaggio di whatsapp viene generato una chiave differente.



Signal vs MTProto

Signal ogni messaggio viene cifrato cun una chiave differente mentre MTProto la chiave viene contrattato ogni 100 messaggi.

Server di Signal non memorizza messaggi ma solo chiavi pubbliche mentre MTProto memorizza i messaggi in chiaro sui server di Telegram.

I messaggi di whatsapp possono essere visualizzati tramite le notifiche.

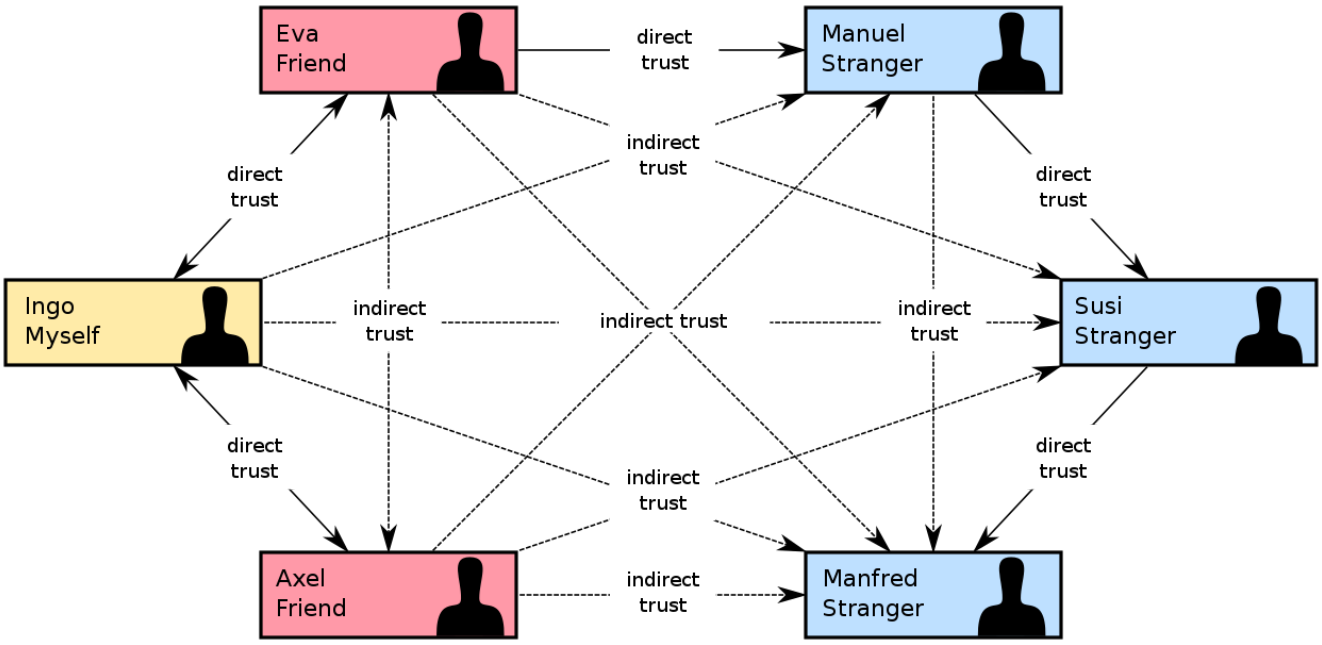
Lab - GPG

OpenPGP

OpenPGP è stato progettato con l'obiettivo di escludere utilizzo di certification authorities centralizzati, ma affidarsi alle relazioni di fiducia definiti dagli utenti regolari. è stato implementato originariamente nel 1992. La sua version open source è stato denominato GNU Privacy Guard GPG. Tutte le distribuzioni Linux utilizzano GPG per la verifica dell'integrità.

Web of trust - OpenPGP

OpenPGP permette di decidere singolarmente di chi fidarsi e quanto.

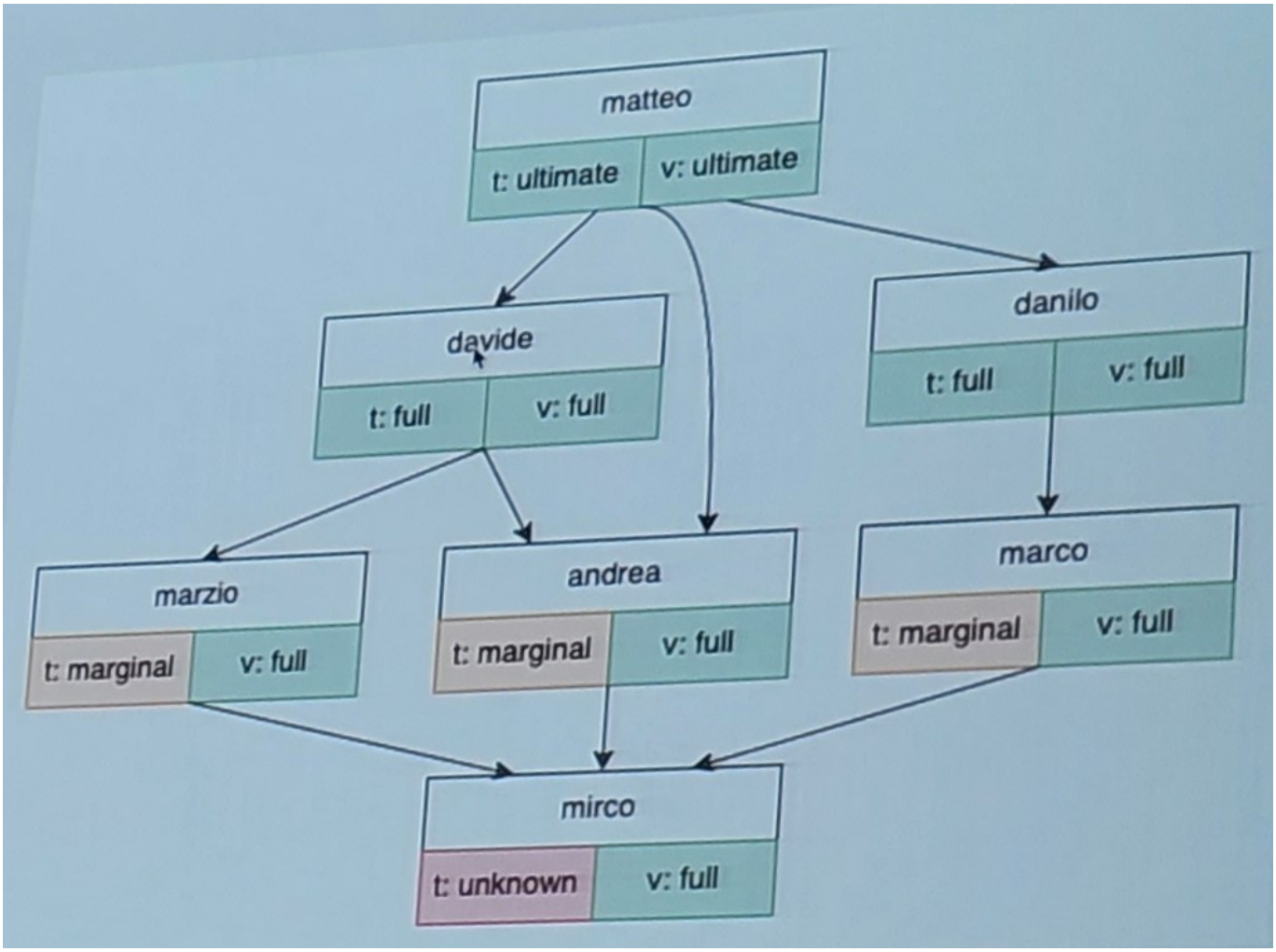



```
docker pull ubuntu
docker run -it ubuntu /bin/bash

gpg --list-keys

# generare una chiave
# tipo di chiave, grandezza della chiave, nome, email, password
gpg --full-generate-key

# aggiungere una chiave, si prende l'hash
# server per caricare la chiave pubblica keyserver.ubuntu.com
gpg --keyserver keyserver.ubuntu.com --recv-keys <hashdellachiave>

# uid unknown la fiducia
gpg --edit-key <nome oppure hash della chiave>

# entro in interfaccia gpg
# trust quanto mi fido della firma

sign # per firmare la chiave scaricato
save # per uscire

trust # permette di cambiare la trust di quella chiave, 1,2,3,...,5
# 5 riservato per la mia chiave
# a questo punto le chiavi firmati dalla chiave scaricata permette di fare trust
sulle chiave trusted

# encryption un file (asimmetrico)
gpg --output file.txt.enc --encrypt --recipient "Nome" file.txt
# decrypting
gpg --output outfile.txt --decrypt file.txt.enc
# encrypting simmetrico
gpg -o file.txt.aes --symmetric --cipher-algo AES256 file.txt
```

```
# decryption simmetrico
gpg -o file.txt.aes -d file.txt

# per firmare
# appende il file con la firma
gpg --output file.txt.sig --sign file.txt

# per firmare senza attaccare il file
gpg --output file.txt.sig --detach-sign file.txt

# per verificare una firma
# e dice info sulla firma
gpg --verify file.txt.sig file.txt

# per esportare la chiave
gpg --export
```

Marginal Trust

Una chiave può diventare marginal se ci sono altre 3 marginal (di cui io ho un trust marginal) che firma quella chiave.



John the ripper

John the ripper è un programma usato decifrare le password partendo da un hash.

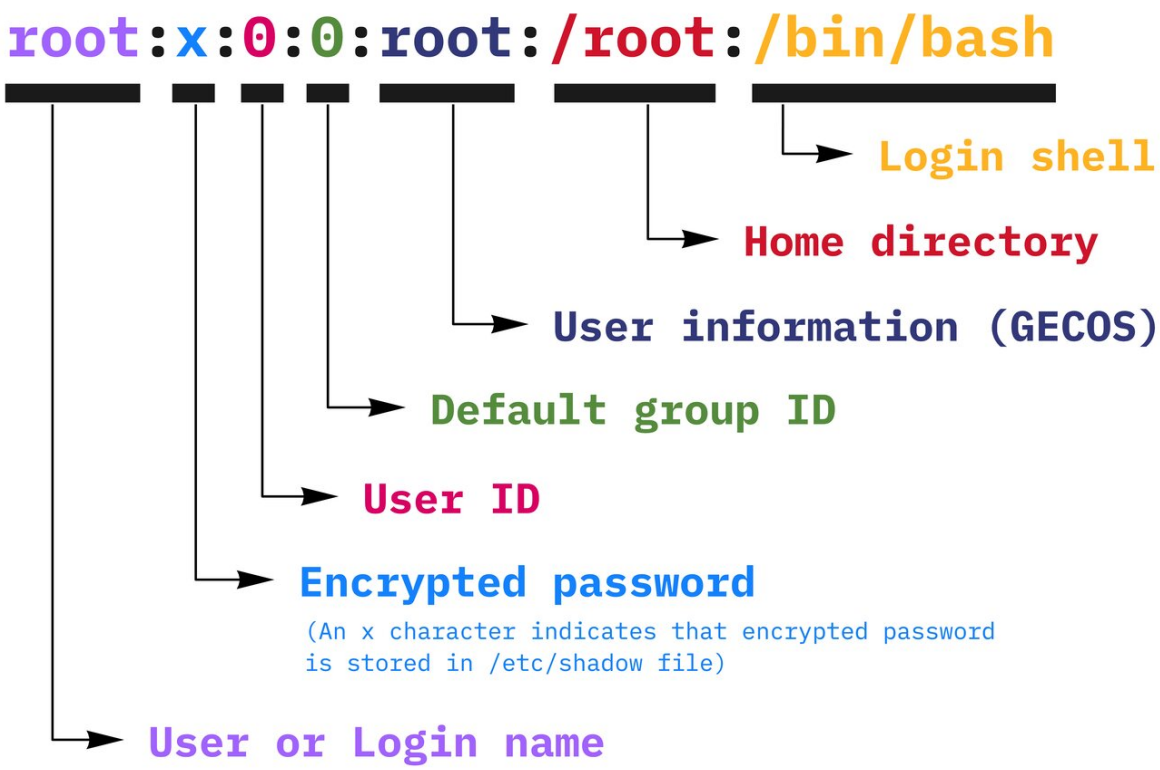
Note: usare la versione `john the ripper` non `john`. [John the Ripper Cheatsheet].

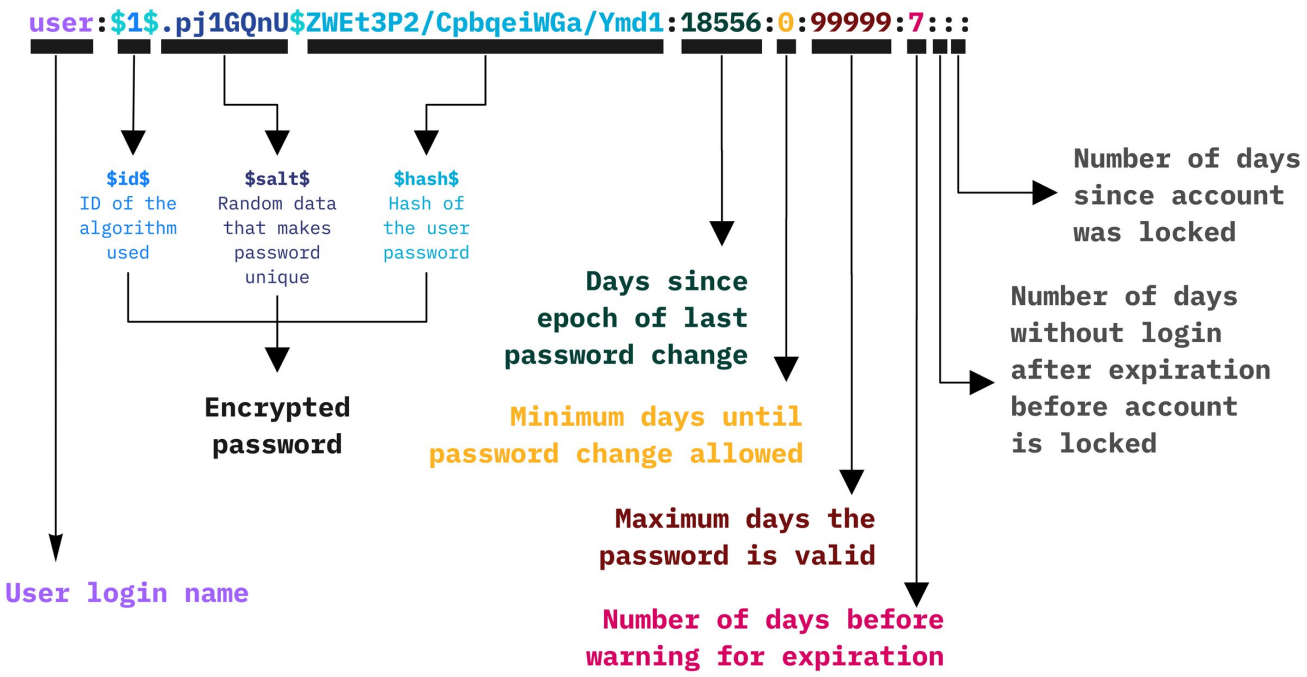
Cracking modes

- Single: `john mypassword --single` usa il campo `gecos` per decifrare il password nome utente, indirizzo, ecc,
- Wordlist: `john mypassword --wordlist=file.txt` usa delle wordlist,
- Incremental: `john --incremental mypassword` è come bruteforce.

`/etc/passwd` & `/etc/shadow`

Il campo `gecos` contiene delle informazioni personali sull'utente.





ID degli algoritmi

- 1 = MD5,
- $2a$ = Blowfish,
- $2y$ = Eksblowfish,
- 5 = SHA256,
- 6 = SHA512.