1 Lezione del 26-11-25

Continuiamo a trattare la sicurezza in rete, in particolare guardando al livello delle applicazioni.

1.1 Sicurezza delle applicazioni

Nello stack protocollare visto finora (livello fisico, datalink, network, trasporto e applicazione) la sicurezza è distibuita a tutti i livelli, cioè implementata ad ogni livello.

Vediamo adesso come questi livelli di sicurezza interagiscono nell'implementazione di un'applicazione. Prendiamo in particolare come esempio un'applicazione di *e-mail* (abbiamo trattato applicazioni di questo tipo in 6.1).

1.1.1 Confidenzialità

Poniamo che Francesca voglia inviare una mail a Maurizio, assicurandone la **confidenzialità**. Abbiamo visto in 23.4 e 23.5 rispettivamente gli approcci di crittografia a chiave **simmetrica** e a chiave **pubblica**.

L'approccio che possiamo pensare di usare è quello di:

1. Generare una chiave di *sessione* K_S, da usare con crittografia a chiave simmetria;
2. Cifrare la chiave di sessione con la chiave pubblica di Maurizio, per ottenere $K_B^+(K_S)$;
3. Cifrare il messaggio vero e proprio, ottenendo $K_S(m)$;
4. A questo punto inviare sia $K_S(m)$ che $K_B^+(K_S)$.

Questo approccio ci permette di ottenere il meglio di entrambi i mondi: la chiave pubblica è più sicura ma più lenta, mentre la chiave simmetrica è più veloce. Criptando la chiave simmetrica con chiave pubblica si assicura la confidenzialità, ma impiegando il carico della crittografia a chiave pubblica su stringhe più brevi (una chiave di alcune centinaaia di bit contro un intero messaggio).

1.1.2 Integrità e autenticazione

Introduciamo nel nostro modello **integrità** dei messaggi e **autenticazione**.

A questo punto possiamo pensare che Francesca dovrà:

1. Firmare digitalmente l'hash del suo messaggio con la sua chiave privata, ottenendo $K_A^-(H(m))$, fornendo come abbiamo visto in 24.1.5 integrità e autenticazione;
2. A questo punto basterà inviare sia il messaggio (in chiaro o criptato), e la firma $K_A^-(H(m))$.

L'approccio ancora migliore, che combina gli ultimi 3 fattori visti, è quello criptare sia la firma che il messaggio, assicurando quindi *confidenzialità* oltre che a *integrità* e *autenticazione*.

Lo schema crittografico che abbiamo descritto finora è sostanzialmente quello della **PGP** (*Pretty Good Privacy*), libreria scritta nel 1991, ad oggi uno dei primi esempi di applicazioni per la sicurezza in rete largamente distribuite.

1.2 TLS

Veniamo quindi al **TLS** (*Transport Layer Security*), un protocollo di sicurezza implementato al di sopra del livello trasporto, supportato dalla maggior parte dei server e browser (gli web server forniscono pagine su TLS sull aporta 443).

TLS fornisce:

- **Confidenzialità**: attraverso crittografia simmetrica;
- **Integrità**: attraverso hashing crittografico;
- **Autenticazione**: attraverso crittografia a chiave pubblica.

1.2.1 Componenti del TLS

Vediamo quelli che sono i componenti di cui abbiamo bisogno per implementare un protocollo TLS:

- **Handshaking**: gli interlocutori dovranno scambiare certificati o chiavi privati per autenticarsi, e scambiarsi o creare un *segreto condiviso*;
- **Derivazione di chiavi**: gli interlocutori useranno il segreto condiviso per derivare una serie di *chiavi* crittografiche. Notiamo che questo è necessario in quanto preferiamo usare crittografia a chiave simmetrica (più efficiente);
- **Trasferimento dati**: i dati verranno inviati in modalità stream, cioè come una serie di record;
- **Closure**: messaggi speciali dovranno essere scambiati per chiudere la connessione.

1.2.2 Handshaking TLS

L'handshaking TLS si svolge a grandi linee come segue:

1. Un interlocutore stabilisce una connessione TCP con l'altro;
2. Segue una fase di autenticazione dove si verifica l'identità dell'altro interlocutore;
3. Quindi si invia una *master secret key* (MS), usata per generare tutte le altre chiavi sulla sessione TLS.

   Si ritiene errato usare la stessa chiave per più funzioni crittografiche, per cui si generano a partire dal MS 4 chiavi:

   - K_C: la chiave di cifratura per i dati che vanno dal client al server;
   - M_C: la chiave MAC per i dati dal client al server;
   - K_S: la chiave di cifratura per i dati che vanno dal server al client;
   - M_S: la chiave MAC per i dati dal server al client.

   Tali chiavi vengono ricavate da una funzione di derivazione (**KDF**, *Key Derivation Function*).

1.2.3 Crittografia dei dati

Abbiamo detto che i dati che viaggiano su sessioni TLS sono suddivisi in **record**. I record TLS suddividano effettivamente l'astrazione del bytestream offerta dal TCP in unità meno granulari, composte dalla tripla criptata:

$$K_C(\texttt{ length : data : mac })$$

ad esempio nel caso di dati inviati da client a server. Queste triple sono quindi composte da:

- La **lunghezza** del record (campo `length`);
- I **dati** veri e propri (campo `data`);
- Il codice **MAC** (campo `mac`), ottenuto in questo caso a partire dalla chiave M_C.

I tipi di attacchi a cui potrebbe sembrare suscettibile questo approccio sono:

- Attacchi di *riordinamento* dei messaggi, dove un man-in-the-middle intercetta segmenti TCP e li riordina (manipolando i numeri di sequenza dell'header TCP, che sono in chiaro);
- Attacchi di *replay*, dove si ripetono record già inviati secondo un procedimento simile al precedente.

Il TLS risolve tali vulnerabilità introducendo *numeri di sequenza TLS*, cioè numeri di sequenza a livello TLS, crittografati e inclusi dentro il MAC.

Una soluzione alternativa è quello di utilizzo del *nonce*, visto in 24.2.

1.2.4 Chiusura di connessione

Un attocco possibile che non abbiamo considerato è il cosiddetto *truncation attack*, dove l'attaccante (per noi Giuseppina) crea un segmento di chiusura TCP fasullo, provocando la terminazione forzata della connessione fra i 2 interlocutori.

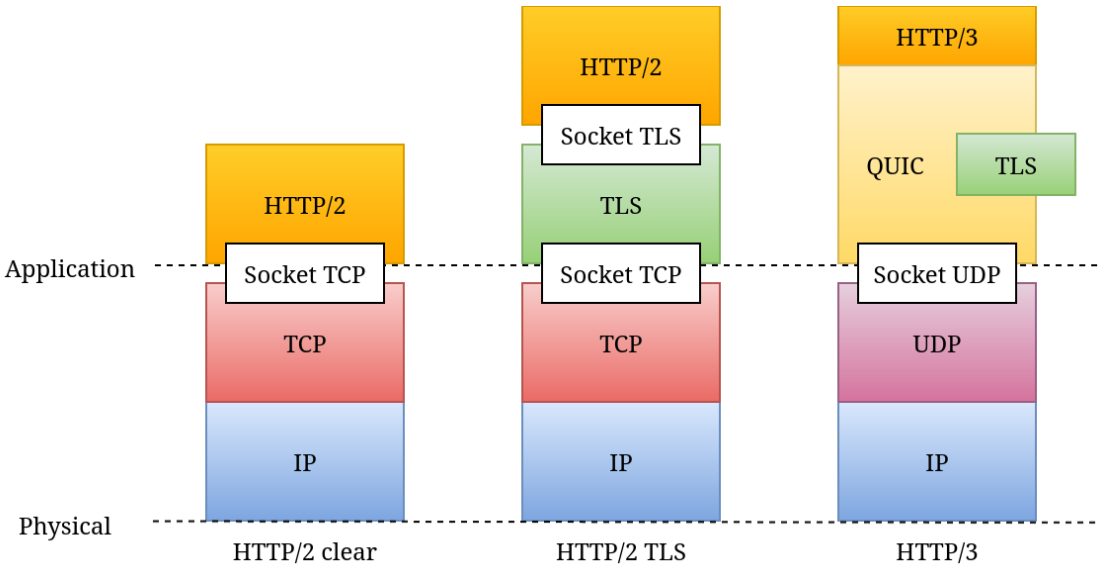
Per risolvere questo problema aggiungiamo alla nostra tripla un campo type, che permette di discriminare fra messaggi normali (0) e messaggi di chiusura(1):

$$K_C(\texttt{ length : type : data : mac })$$

Questo signfica che la connessione TCP non può essere chiusa finché non si chiude la connessione TLS soprastante, dove le richieste di terminazinoe avvengono in maniera criptata.

1.2.5 Posizionamento del TLS

Può essere interessante vedere dove il TLS si posiziona a livello di stack protocollare, in 3 configurazioni diverse (*HTTP/2 clear*, cioè senza TLS, *HTTP/2* normale con TLS, e *HTTP/3*, che abbiamo introdotto in 5.2 e approfondito in 22.2):



- Per quanto riguarda **HTTP/2 clear**, abbiamo che il TLS chiaramente non è presente, e quindi il protocollo HTTP a livello application si svolge interamente sopra il socket TCP;
- Se dotiamo l'**HTTP/2** del TLS, questo va a inserirsi sempre al livello application, ma al di sotto del protocollo HTTP stesso. Si a quindi che è il TLS ad interagire con il socket TCP, offrendo un socket TLS al livello HTTP soprastante;
- Infine vediamo, solo a scopo di esempio, come la stessa semantica è implementata in **HTTP/3** su QUIC: in questo caso si ha che il protocollo di livello trasporto utilizzato è l'UDP, al di sopra del cui è il QUIC a offrire servizio di trasporto affidabile all'HTTP/3. Il TLS risulta implementato direttamente dentro QUIC.

1.2.6 Suite di cifratura

La **suite** di cifratura è formata da un insieme di algoritmi crittografici e per la generazione di chiavi, cioè:

- Un algoritmo per la generazione di chiavi;
- Un algoritmo di crittografia a chiave pubblica;
- Un algoritmo di crittografia a chiave simmetrica;
- Un algoritmo di MAC.

TLS 1.3 (del 2018) fornisce 5 scelte per le suite, mentre il vecchio TLS 1.2 (del 2008) ne forniva ben 37.

Il problema che però ci interessa è come avviene la scelta e la comunicazione della suite scelta fra host in comunicazione.

1. Prevederemo quindi in fase di handshake che il client comunichi al server quali suite di cifratura è capace di usare;
2. Il server risponderà inviando la suite di cifratura scelta fra quelle supportate dal client. Seguirà il certificato del server.

3. Il client potrà quindi controllare il certificato server ricevuto, generare le chiavi attraverso la suite scelta, e iniziare ad effettuare richieste di livello application.

1.3 IP Sec

IP Sec (da *Internet Protocol Security*) è una suite di protocolli di sicurezza che lavorano al livello IP, permettendo crittografia, autenticazione ed integrità a livello datagramma.

In questo, IP Sec riguarda sia il traffico utente che quello di controllo (DNS, ecc...).

IP Sec opera in 2 modalità:

- Modalità **trasporto**: solo i payload dei *datagrammi* vengono crittografati e autenticati;
- Modalità **tunnel**: l'intero datagramma viene crittografato e autenticato, e quindi incapsulato in un nuovo datagramma con un nuovo header IP, che quindi *"tunnellizza"* il vecchio datagramma.

1.3.1 IP Sec e VPN

Spesso le istituzioni vogliono sfruttare reti private per ragioni di sicurezza. Questo però risulta molto dispendioso in termini di costi di infrastruttura.

Sfruttando una **VPN** (*Virtual Private Network*), il traffico di un'istituzione può invece viaggiare sull'internet pubblico (dopo essere stato criptato).

L'idea del VPN è quindi quello di creare reti effettivamente **private**, ma il cui traffico viaggia interamente all'interno dell'internet *pubblico*, ma in forma criptata. Questo permette ad esempio ad *esterni* di connettersi ad una rete privata da remoto: il traffico sarebbe comunque viaggiato sull'internet pubblico.

1.3.2 Estensioni di IP

Per realizzare queste funzionalità dobbiamo quindi estendere il protocollo IP.Esistono due protocolli principali in questo merito:

- Protocollo **AH** (*Authentication Header*): fornisce autenticazione e integrità dati, ma non confidenzialità;
- Protocollo **ESP** (*Encapsulation Security Protocol*): fornisce autenticazione, integrità dati e confidenzialità. Sarà;

Estendere l'IP significa effettivamente renderlo *stateful*: dobbiamo realizzare delle **SA** (*Security Associations*) fra *endpoint* (solitamente router) prima di effettuare le trasmissioni vere e proprie. Queste associazioni sono *direzionali* (dal trasmettitore al ricevitore). Abbiamo quindi che gli endpoint mantengono informazioni riguardo allo *stato* dell'SA (da cui *stateful*).

Vediamo nel dettaglio quali lqueste informazioni possono essere:

- Un identificatore su 32 bit, detto **SPI** (*Security Parameter Index*);
- L'interfaccia SA di origine (quella del trasmettitore);
- L'interfaccia SA di destinazione (quella del ricevitore);
- Il tipo di crittografia usato;

- La chiave crittografica usata;
- Il tipo di controllo di integrità usato;
- La chiave di autenticazione usata.

1.3.3 Datagramma IP Sec

Un datagramma ESP è incluso all'interno di un datagramma IP (in *tunneling*). Presenta quindi i seguenti campi:

```
new IP header
ESP header
  - SPI
  - Seq#
orig. IP header
orig. IP payload
ESP trailer
  - padding
  - pad length
  - next header
ESP auth
```

- Il **trailer ESP** rapprsenta padding per gli algoritmi di cifratura a blocchi;
- L'**header ESP** contiene l'SPI, così che il ricevitore possa elaborare il datagramma (gestendo le sue SA), e il numero di sequenza (per evitare attacchi di replay);
- L'**auth ESP** contiene il MAC dell'intero datagramma ESP.