Introduzione TLS pdf20

2 dicembre 2025

Indice

1 Duplice obiettivo della lezione

→ **Dissezione di un protocollo di sicurezza ben noto e ampiamente utilizzato**

⇒ Analisi ragionevolmente approfondita dei dettagli del TLS

→ Il diavolo sta nei dettagli :-)

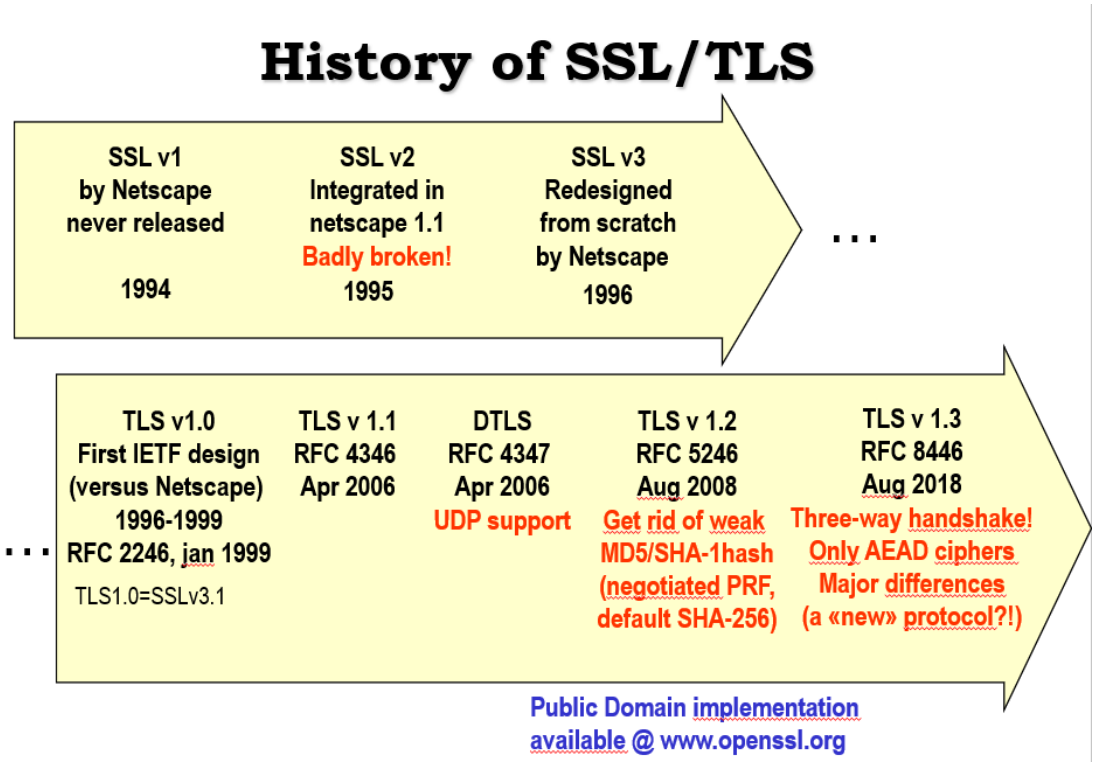
→ Sebbene molti altri dettagli siano stati omessi (questo non è un corso completo su TLS. . .)

→ **Capire come dovrebbe essere progettato un protocollo di sicurezza longevo**

⇒ Mostrare scelte di progettazione buone vs cattive

→ Il TLS mostra diversi esempi per entrambe!

2 Storia ed Evoluzione di SSL/TLS



L'evoluzione dei protocolli di sicurezza per il web si divide in due grandi ere: quella iniziale dominata da Netscape con il protocollo SSL, e quella successiva standardizzata dalla IETF con il protocollo TLS.

2.1 L'era SSL (Netscape)

Nei primi anni '90, Netscape ha sviluppato i primi tentativi di crittografia per il web.

SSL v1 (1994) Sviluppato da Netscape, questo protocollo non è **mai stato rilasciato** pubblicamente a causa di gravi difetti di sicurezza presenti fin dal principio.

SSL v2 (1995) Integrato in Netscape 1.1, rappresenta il primo rilascio pubblico. Tuttavia, si rivelò presto **gravemente compromesso** (*Badly broken!*) dal punto di vista della sicurezza, rendendolo inadeguato.

SSL v3 (1996) Per rimediare ai fallimenti precedenti, Netscape riprogettò il protocollo da zero (*Redesigned from scratch*). Questa versione ha costituito la base per il futuro standard TLS.

2.2 L'era TLS (Standard IETF)

Successivamente, lo sviluppo passò sotto il controllo della IETF (*Internet Engineering Task Force*), garantendo uno standard aperto e non proprietario.

TLS v1.0 (Gennaio 1999 - RFC 2246) Rappresenta il primo design ufficiale IETF, anche se tecnicamente è molto simile al suo predecessore (spesso indicato come SSLv3.1). È stato lo standard dominante per molti anni (1996-1999).

TLS v1.1 (Aprile 2006 - RFC 4346) Un aggiornamento incrementale che ha introdotto protezioni contro attacchi specifici (come le vulnerabilità sui vettori di inizializzazione).

DTLS (Aprile 2006 - RFC 4347) Parallelamente, è stato introdotto il *Datagram TLS*, progettato specificamente per fornire **supporto UDP**, essenziale per applicazioni in tempo reale (come lo streaming o il VoIP) che non usano TCP.

TLS v1.2 (Agosto 2008 - RFC 5246) Un passo avanti significativo nella sicurezza crittografica:

- Eliminazione delle funzioni di hash deboli (MD5/SHA-1).
- Introduzione della negoziazione per la funzione pseudo-casuale (PRF).
- Adozione di **SHA-256** come default.

TLS v1.3 (Agosto 2018 - RFC 8446) L'ultima e più radicale evoluzione, tanto da essere considerata quasi un **"nuovo protocollo"**:

- Semplificazione drastica: supporto esclusivo per cifrari **AEAD** (Authenticated Encryption with Associated Data).
- Prestazioni migliorate grazie a un **Handshake a tre vie** ottimizzato.
- Rimozione di vecchie funzionalità insicure per ridurre la superficie di attacco.

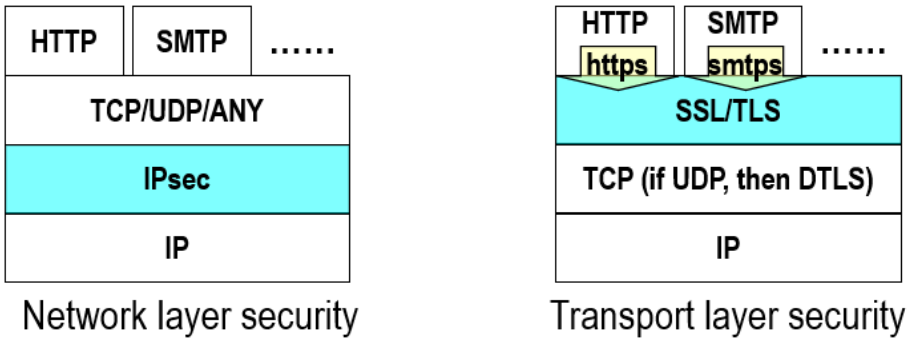
Nota: Un'implementazione di riferimento di pubblico dominio di questi protocolli è disponibile tramite il progetto `OpenSSL` (www.openssl.org).

3 Posizionamento nello Stack di Rete (Layered View)

Per comprendere il funzionamento di SSL/TLS, è fondamentale visualizzare dove si colloca all'interno della pila dei protocolli di rete. La slide propone un confronto tra la sicurezza a livello di rete e quella a livello di trasporto.

3.1 Confronto Architetturale

Esistono due approcci principali per proteggere le comunicazioni:



- **Network Layer Security (IPsec):** Come mostrato nel diagramma di sinistra, la sicurezza viene applicata direttamente sopra il protocollo IP. In questo scenario (tipico delle VPN basate su IPsec), il livello di sicurezza è trasparente ai livelli superiori: TCP, UDP e le applicazioni (HTTP, SMTP) poggiano su un livello IP già sicuro.
- **Transport Layer Security (SSL/TLS):** Nel diagramma di destra, l'approccio cambia. SSL/TLS si inserisce come uno strato intermedio. Le applicazioni non comunicano più direttamente con il livello di trasporto (TCP), ma passano attraverso lo strato SSL/TLS. Di conseguenza, i protocolli applicativi cambiano nomenclatura (HTTP diventa `https`, SMTP diventa `smtps`), indicando l'incapsulamento sicuro.

3.2 Caratteristiche del posizionamento SSL/TLS

Il protocollo SSL/TLS possiede caratteristiche uniche riguardo la sua collocazione:

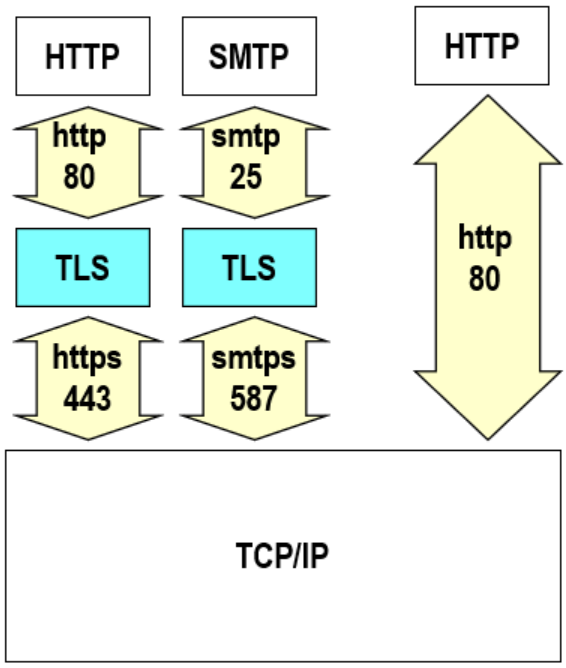
1. **Sopra TCP, ma sotto il livello Applicazione** SSL/TLS non fa parte del kernel del sistema operativo (come TCP/IP), ma opera nello spazio utente.
   - Può essere concettualmente visto come il *sottolivello più alto* del Livello 4 (Trasporto) oppure come il *sottolivello più basso* del Livello 7 (Applicazione).
   - **Nota Importante:** SSL/TLS non è un miglioramento di sicurezza del TCP stesso. Il TCP rimane invariato; TLS utilizza il TCP semplicemente come canale di trasporto affidabile su cui costruire il proprio handshake e la cifratura dei dati. Se il trasporto è UDP, si utilizza la variante DTLS.
2. **Indipendenza dal livello di Trasporto Internet (L4)** Sebbene sia nato per il web, SSL/TLS non è limitato strettamente al livello 4 dello stack TCP/IP. È progettato per gestire relazioni *point-to-point* in generale.
   - Un esempio lampante è **EAP-TLS** (RFC 2716). In questo caso, il protocollo TLS viene utilizzato per l'autenticazione e la protezione dell'integrità direttamente sopra il livello 2 (Data Link), all'interno del protocollo EAP (spesso usato nelle reti Wi-Fi Enterprise).

4 Supporto Applicativo e Gestione delle Porte

Quando si introduce la crittografia TLS in protocolli applicativi esistenti (come HTTP o SMTP), sorge una scelta architetturale fondamentale: utilizzare porte di rete dedicate o negoziare la sicurezza sulla porta standard? La slide analizza due approcci distinti.

4.1 Approccio Storico: Porte Dedicate

L'idea iniziale, definita nella slide come una *"cattiva idea storica"*, consisteva nel riservare un numero di porta speciale per la versione protetta (SSL/TLS) di ogni protocollo. Questo crea una separazione netta:



- **HTTP:** Porta 80 (chiaro) → **HTTPS:** Porta 443 (sicuro)

Questa logica si è estesa rapidamente ad altri servizi, portando a una duplicazione delle porte necessarie:

- **SMTP (email):** Porta 25 → **SMTPS:** Porta 465 (implementazione Microsoft) o 587 (altri).
- **POP3:** Porta 110 → **POP3S:** Porta 995.
- **IMAP:** Porta 143 → **IMAPS:** Porta 991.
- **Telnet:** Porta 23 → **TelnetS:** Porta 992.

Analisi Costi/Benefici

Vantaggi (Pros):
- Funziona bene ed è diventato lo *standard de facto*.

- Il supporto lato applicazione è semplice ("Straightforward application support"): se arriva traffico sulla porta 443, il server sa immediatamente che deve aspettarsi un handshake TLS.

Svantaggi (Cons):

- Richiede **due numeri di porta riservati** per lo stesso identico servizio.
- Questo approccio è stato tecnicamente **deprecato dalla IETF**, nonostante sia ancora onnipresente.

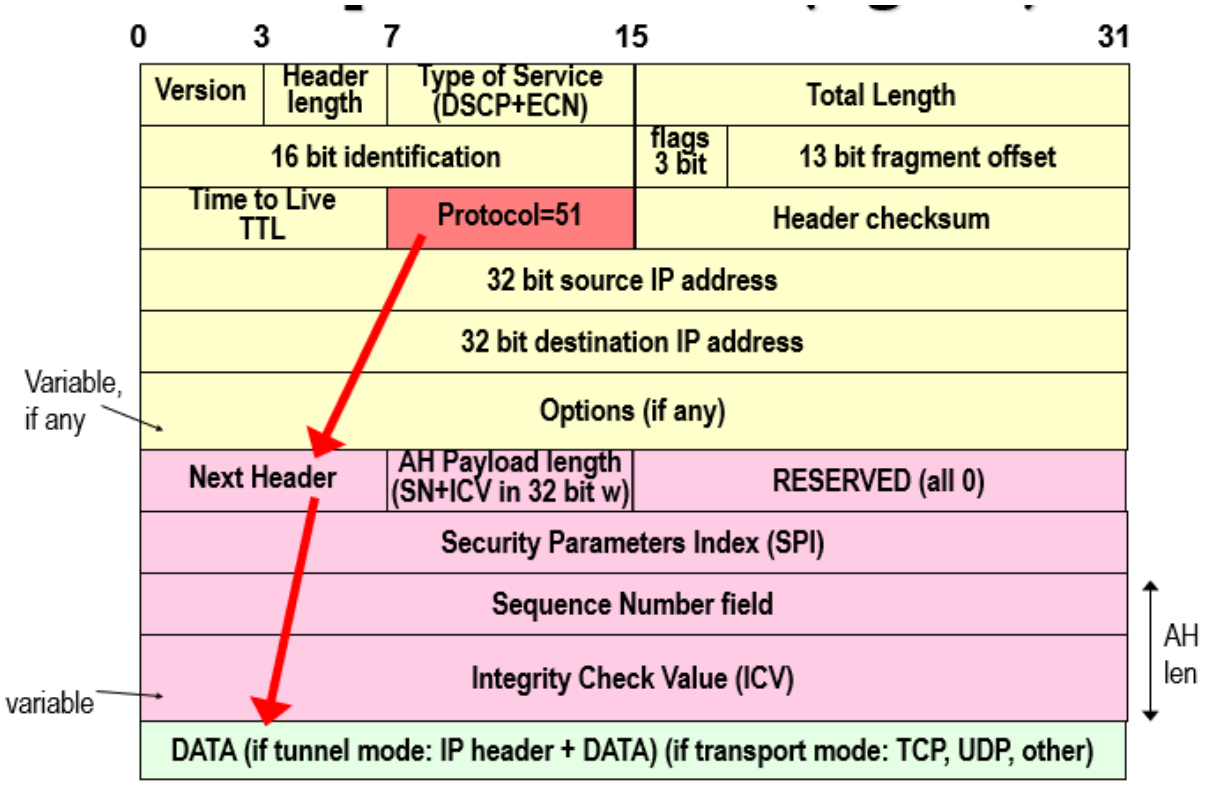
4.2 Approccio Alternativo: Adattamento Interno (Upgrade)

L'approccio moderno e raccomandato consiste nell'adattare leggermente le "interne" dell'applicazione per riutilizzare la **stessa porta**.

- L'applicazione parte in chiaro sulla porta standard.
- Il client invia un comando speciale per richiedere il passaggio alla crittografia.
- **Esempio (RFC 2817):** In HTTPv1.1, il client può inviare l'header `Upgrade: TLS/1.0`. Se il server accetta, la connessione esistente sulla porta 80 passa da testo in chiaro a cifrata (TLS).

Visivamente (come mostrato nel diagramma a destra della slide), questo significa che non c'è più una "scatola" TLS separata su una porta diversa, ma il protocollo HTTP stesso gestisce la transizione all'interno dello stesso canale di comunicazione TCP/IP.

5 Struttura del Pacchetto IPsec: Authentication Header (AH)



La figura illustra come un pacchetto IPv4 standard viene modificato per implementare la sicurezza tramite il protocollo IPsec *Authentication Header* (AH). La struttura è composta da tre sezioni principali impilate, collegate logica dal meccanismo dei "Next Header".

5.1 1. Intestazione IPv4 (Blocco Giallo)

Questa è l'intestazione IP standard (lunghezza variabile, tipicamente 20 byte).

- La modifica cruciale rispetto a un pacchetto normale risiede nel campo **Protocol** (evidenziato in rosso).
- Il valore è impostato a **51**. Questo numero segnala al destinatario che il contenuto immediatamente successivo non è un protocollo di trasporto (come TCP o UDP), ma un'intestazione *Authentication Header* (AH).
- Gli altri campi (Version, TTL, Source/Destination IP) rimangono quelli standard dell'IPv4.

5.2 2. Intestazione AH (Blocco Rosa)

L'intestazione AH viene inserita come uno "strato intermedio" (shim header) subito dopo l'intestazione IP. Contiene i campi necessari per garantire l'autenticazione e l'integrità del pacchetto:

Next Header (8 bit): Identifica il tipo di dati contenuto nel payload successivo (il blocco verde). Ad esempio, se il payload è TCP, questo campo varrà 6. Le frecce rosse nello schema indicano questa catena logica: *IP Header (Proto=51)* $\rightarrow$ *AH Header (Next=TCP)* $\rightarrow$ *Dati TCP*.

Payload Length: Indica la lunghezza dell'intestazione AH stessa.

RESERVED: Campo riservato (impostato a 0) per usi futuri.

Security Parameters Index (SPI): Un valore arbitrario di 32 bit che, combinato con l'indirizzo IP di destinazione e il protocollo (AH), permette al ricevente di identificare la *Security Association* (SA) corretta da utilizzare per decifrare/verificare il pacchetto.

Sequence Number: Un contatore crescente di 32 bit utilizzato per proteggere dagli attacchi di tipo *Replay* (reiniezione di pacchetti vecchi).

Integrity Check Value (ICV): Contiene il valore di hash (o firma digitale) calcolato sul pacchetto. È di lunghezza variabile a seconda dell'algoritmo usato (es. HMAC-SHA1, HMAC-MD5) ed è il campo che garantisce l'integrità dei dati.

5.3 3. Dati / Payload (Blocco Verde)

Questa è la parte finale del pacchetto, che contiene il carico utile vero e proprio. Il suo contenuto dipende dalla modalità IPsec utilizzata:

- **Transport Mode:** Contiene direttamente il segmento del livello di trasporto (es. intestazione TCP/UDP + dati dell'applicazione).
- **Tunnel Mode:** Contiene l'intero pacchetto IP originale (Intestazione IP originale + Dati). In questo caso, l'intestazione IP nel blocco giallo è quella del gateway del tunnel, mentre quella nel blocco verde è quella del mittente originale.

6 Obiettivi e Filosofia del TLS

Il protocollo TLS è progettato per soddisfare due obiettivi primari, che corrispondono a due fasi distinte della comunicazione sicura. A differenza di altri protocolli che separano nettamente queste funzioni, il TLS adotta un approccio integrato ("Two-in-One").

6.1 1. Stabilire una Sessione (Fase di Handshake)

Prima di inviare qualsiasi dato applicativo reale, il TLS deve instaurare un contesto sicuro. Questa fase, nota come **TLS Handshake**, ha tre scopi fondamentali:

- **Negoziazione degli Algoritmi (Agree on algorithms):** Client e Server devono accordarsi su quale "lingua" crittografica parlare. Questo set di algoritmi (per lo scambio chiavi, la cifratura e l'hashing) viene chiamato *Cipher Suite*.
- **Condivisione dei Segreti (Share secrets):** Le parti devono generare e scambiarsi il materiale crittografico (chiavi) che verrà utilizzato per cifrare la connessione.
- **Autenticazione (Perform authentication):** Verifica dell'identità delle parti comunicanti (solitamente il server tramite certificato digitale, e opzionalmente il client).

6.2 2. Trasferimento dei Dati Applicativi

Una volta completato l'handshake, inizia la trasmissione vera e propria dei dati (il livello *Record Protocol*). Qui gli obiettivi sono:

- **Privacy della Comunicazione:** Garantita tramite **cifratura simmetrica** (molto più veloce di quella asimmetrica usata nell'handshake). Questo impedisce l'intercettazione (eavesdropping).
- **Integrità dei Dati:** Garantita tramite **HMAC** (Keyed-Hash Message Authentication Code). Questo assicura che i dati non siano stati manomessi durante il transito; ogni pacchetto ha una "firma" che ne valida il contenuto.

6.3 Approccio Architetturale: "Two-in-One"

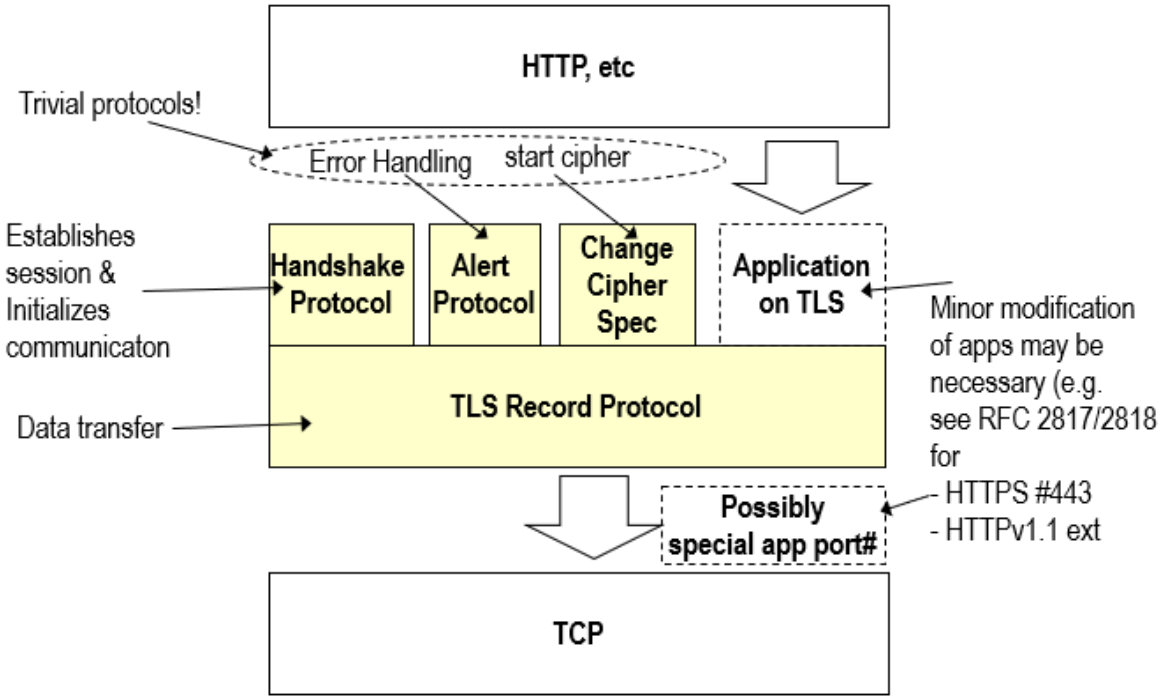
La slide evidenzia una caratteristica distintiva del TLS rispetto ad altri protocolli di sicurezza Internet (come IPsec).

Altri Protocolli (es. IPsec): Tendono a distinguere nettamente il protocollo per stabilire la sessione da quello per il trasporto dati.

- *Esempio:* In IPsec, **IKE** (Internet Key Exchange) gestisce le chiavi, mentre **ESP/AH** gestiscono il trasporto sicuro dei pacchetti. Sono moduli separati.

Protocollo TLS: È un pacchetto "Tutto in uno". Definisce al suo interno sia le regole per l'handshake che quelle per il framing e la protezione dei dati (Record Layer). Sebbene siano concettualmente fasi diverse, sono specificate all'interno dello stesso standard monolitico.

7 Architettura dello Stack TLS



Il protocollo TLS è strutturato internamente in due livelli principali: un livello inferiore di trasporto dati e un livello superiore di gestione e supervisione.

7.1 1. TLS Record Protocol (Livello Inferiore)

Alla base dello stack TLS (il blocco giallo grande in basso) troviamo il **TLS Record Protocol**.

- **Funzione:** È il "trasportatore" universale. Si occupa di frammentare, comprimere (opzionalmente), cifrare e garantire l'integrità dei dati (tramite MAC).
- **Posizione:** Risiede direttamente sopra il protocollo di trasporto (TCP).
- **Compito:** Fornisce servizi di sicurezza di base ai protocolli di livello superiore. Tutto ciò che viene generato dai protocolli sovrastanti viene incapsulato dentro pacchetti del Record Protocol per essere trasmesso.

7.2 2. Protocolli di Gestione (Livello Superiore)

Sopra il Record Protocol operano tre protocolli specifici di gestione del TLS (più i dati dell'applicazione stessa). La slide distingue tra la complessità dell'Handshake e la semplicità degli altri due.

Handshake Protocol: È il componente più complesso e sofisticato.

- Serve a stabilire la sessione e inizializzare la comunicazione.
- Qui avviene l'autenticazione del server (e del client), la negoziazione degli algoritmi e lo scambio delle chiavi.

Change Cipher Spec Protocol: Definito nella slide come un "protocollo banale" (*Trivial protocol*).

- Ha un unico compito: segnalare la transizione (*start cipher*).
- Indica al ricevente che, da quel momento in poi, i messaggi successivi saranno cifrati con le chiavi e gli algoritmi appena negoziati.

Alert Protocol: Anch'esso considerato un "protocollo banale".

- Gestisce la segnalazione degli errori (*Error Handling*).
- Può inviare avvisi di chiusura sessione (warning) o errori fatali che interrompono la connessione (es. fallimento della verifica del certificato).

7.3 3. Interfaccia con le Applicazioni

Il quarto elemento che poggia sul Record Protocol è l'**Application on TLS** (es. HTTP).

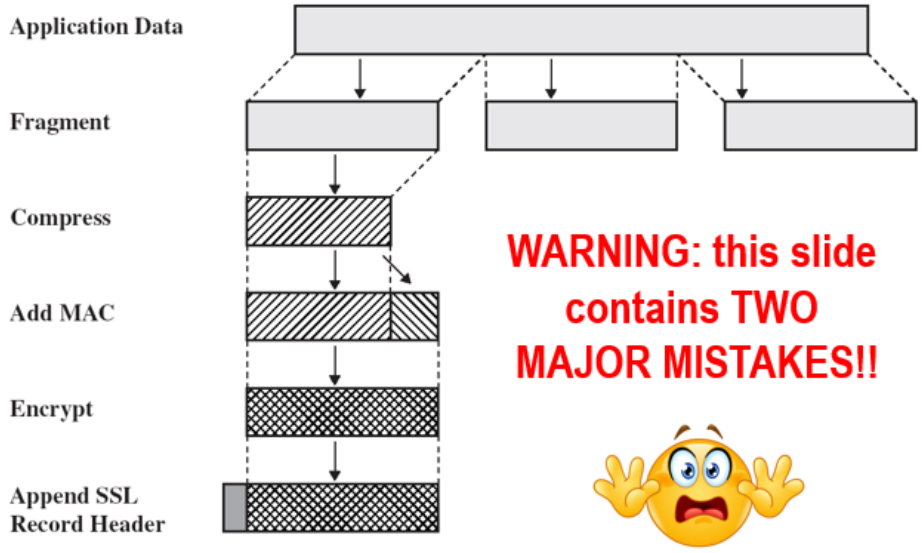
- Le applicazioni (come i browser web) inviano i loro dati al TLS invece che direttamente al TCP.
- Come indicato nella slide, questo richiede solitamente **"modifiche minori"** alle applicazioni (es. supporto RFC 2817/2818) o l'uso di porte specifiche (come la porta 443 per HTTPS) per instradare correttamente il traffico attraverso lo stack di sicurezza.

Sintesi visiva: Il flusso dei dati scende dall'applicazione (HTTP), viene gestito dai protocolli di handshake/controllo se necessario, viene impacchettato e cifrato dal *Record Protocol*, e infine affidato al TCP per la trasmissione fisica sulla rete.

8 Operazioni del TLS Record Protocol (e le sue criticità)

Questa sezione illustra la pipeline di elaborazione dei dati nel protocollo TLS (specifico per le versioni fino alla 1.2). Tuttavia, come evidenziato dall'avvertimento nella slide, questo specifico design nasconde due gravi difetti di progettazione che sono stati corretti solo nelle versioni moderne (TLS 1.3).

8.1 Record Protocol operation - Il Flusso di Elaborazione (Pipeline)



Quando un'applicazione invia dei dati, questi attraversano le seguenti fasi prima di essere trasmessi in rete:

1. **Frammentazione (Fragment):** I dati dell'applicazione vengono spezzati in blocchi gestibili. La dimensione massima di un frammento è di 2^{14} byte (16 KB).
2. **Compressione (Compress):** Il frammento viene (opzionalmente) compresso per ridurre la larghezza di banda utilizzata.
3. **Aggiunta del MAC (Add MAC):** Viene calcolato un codice di autenticazione (HMAC) sui dati compressi. Questo serve a garantire l'integrità.
4. **Cifratura (Encrypt):** I dati compressi *più* il MAC vengono cifrati insieme. Questo schema è noto come **MAC-then-Encrypt** (prima calcolo il MAC, poi cifro tutto).
5. **Aggiunta Header (Append Header):** Viene aggiunto l'intestazione del Record Protocol (che contiene tipo di contenuto, versione e lunghezza) all'inizio del pacchetto cifrato.

8.2 I "Due Grandi Errori" (Analisi dell'Avvertimento)

La slide riporta un avviso allarmante: *"This slide contains TWO MAJOR MISTAKES"*. Questi non sono errori nel disegno (che rappresenta fedelmente come funzionava TLS 1.2), ma errori concettuali di sicurezza scoperti nel tempo:

Errore 1: La Compressione. Sebbene sembri una buona idea per le prestazioni, la compressione a livello TLS apre la porta ad attacchi "Side-Channel" (canale laterale), come **CRIME** e **BREACH**.

- *Il problema:* Un attaccante può iniettare dati nel flusso e osservare di quanto cambia la lunghezza del pacchetto cifrato finale. Poiché la compressione riduce i dati ripetuti, analizzando la lunghezza del risultato l'attaccante può indovinare parti del segreto (come i Cookie di sessione).

- *Soluzione TLS 1.3:* La compressione è stata completamente rimossa.

Errore 2: L'ordine MAC-then-Encrypt. L'ordine delle operazioni mostrato (calcolare il MAC e poi cifrare) si è rivelato vulnerabile.

- *Il problema:* Questo approccio espone a "Padding Oracle Attacks" (come l'attacco *Lucky13*). Il server, quando decifra, deve prima verificare il padding e poi il MAC. Le differenze minime di tempo impiegate per questi due errori permettono a un attaccante di decifrare il contenuto.
- *Soluzione Moderna:* Si utilizza l'approccio **Encrypt-then-MAC** (usato in IPsec) o, ancora meglio, la cifratura autenticata **AEAD** (obbligatoria in TLS 1.3), che esegue cifratura e integrità in un unico passaggio atomico sicuro.

SECURITY FOCUS: Attacchi Padding Oracle (MAC-then-Encrypt)

L'errore architetturale del **MAC-then-Encrypt** risiede nel fatto che il ricevente deve prima decifrare il pacchetto, controllare il padding e, solo successivamente, verificare il MAC. Se un server risponde in modo diverso (ad esempio con messaggi di errore differenti o con tempi di risposta misurabilmente diversi) nel caso di "Padding Errato" rispetto al caso di "MAC Errato", esso agisce come un **Oracolo**. Un attaccante può intercettare un pacchetto cifrato e inviarne migliaia di varianti leggermente modificate. Analizzando le risposte dell'"Oracolo" (il server), l'attaccante può dedurre il contenuto del messaggio originale byte per byte, violando la confidenzialità.

9 Fase di Frammentazione (TLS Fragmentation)

La frammentazione è la prima operazione compiuta dal TLS Record Protocol quando riceve dati dai livelli superiori. Si verifica esattamente all'interfaccia tra l'Applicazione e il TLS.

9.1 Differenza tra Frammentazione TLS e Segmentazione TCP

La slide pone un accento critico su una distinzione fondamentale per evitare confusione terminologica e concettuale:

Frammentazione TLS (Livello 6/5): È un'operazione logica. Il TLS divide i dati in "pezzi" gestibili per poter applicare le operazioni crittografiche (MAC e cifratura) su blocchi finiti.

Segmentazione TCP (Livello 4): È un'operazione di trasporto. Il TCP divide i dati in segmenti in base alla *Maximum Segment Size* (MSS) per adattarsi ai limiti fisici della rete (MTU).

Nota Bene: La frammentazione TLS avviene *prima* che i dati vengano passati al TCP. Un singolo frammento TLS potrebbe essere successivamente diviso in più segmenti TCP, oppure più frammenti TLS piccoli potrebbero finire in un unico segmento TCP.

9.2 Gestione dell'Input

Il protocollo è progettato per gestire flussi di dati di qualsiasi natura:

- **Input di dimensione arbitraria:** L'applicazione può inviare al TLS un flusso continuo di dati (stream) o blocchi enormi. Il TLS si occupa di "bufferizzarli" e tagliarli.
- **Aggregazione:** Se l'applicazione invia messaggi molto piccoli e frequenti dello **stesso protocollo** (es. più messaggi di Handshake ravvicinati), il TLS può aggregarli in un unico frammento per efficienza, riducendo l'overhead delle intestazioni.

9.3 Dimensione del Frammento

La specifica definisce rigidamente la dimensione massima del testo in chiaro (*plaintext*) che può essere contenuto in un record:

$$\text{Max Fragment Size} = 2^{14} \text{ bytes} = 16384 \text{ bytes}$$

Questo significa che ogni "record" TLS non può trasportare più di 16KB di dati utili. Se l'applicazione invia un file da 1MB, questo verrà spezzato in circa 64 frammenti da 16KB ciascuno.

- Questo limite è importante perché il ricevente deve bufferizzare l'intero frammento per verificarne l'integrità (calcolare il MAC) prima di poterlo passare all'applicazione. Frammenti troppo grandi causerebbero latenza.

10 La Compressione in TLS: Ascesa e Declino

La compressione dei dati (ovviamente *lossless*, senza perdita di informazione) è stata per lungo tempo una caratteristica controversa all'interno dei protocolli SSL/TLS. L'obiettivo era ridurre la quantità di dati da cifrare e trasmettere, migliorando le prestazioni.

10.1 Cronistoria dell'implementazione

- **Le origini (SSLv2):** La compressione era presente e utilizzata nelle prime versioni del protocollo (SSLv2).
- **La pausa (SSLv3 / TLSv1.0):** Con l'arrivo degli standard successivi, la compressione fu formalmente considerata ma non specificata. Di default, il metodo di compressione era impostato a `null` (nessuna compressione).
- **Il ritorno (2004):** Con la RFC 3749, la IETF reintrodusse ufficialmente la compressione definendo l'algoritmo **DEFLATE** (basato su RFC 2951). Successivamente, la RFC 3943 aggiunse il supporto per l'algoritmo Lempel-Ziv-Stac (LZS).

10.2 Il picco di popolarità (Early 2012)

Verso l'inizio degli anni 2010, la compressione divenne estremamente attraente.

- **Motivazione:** Il web stava cambiando. L'uso massiccio di formati "verbosi" basati su testo, come **XML** (usato in SOAP/AJAX) e HTML sempre più complessi, rendeva la compressione un ottimo strumento per risparmiare banda e latenza.
- **Adozione:** All'inizio del 2012, le statistiche mostravano una diffusione massiccia:
  - Circa il **45% dei browser** la supportava (inclusi Chrome e Firefox).
  - Circa il **42% dei server** la supportava.

10.3 La "Morte Improvvisa" (Settembre 2012)

Nonostante la popolarità, la compressione a livello TLS è stata bruscamente abbandonata ("Suddenly died") nel settembre 2012.

L'attacco CRIME: I ricercatori Juliano Rizzo e Thai Duong dimostrarono che la compressione permetteva di violare la segretezza del canale tramite un attacco *Side-Channel* chiamato **CRIME** (Compression Ratio Info-leak Made Easy).

Il problema: Se un attaccante può iniettare dati nel flusso (es. tramite JavaScript in un browser), può osservare quanto bene il protocollo comprime il pacchetto. Poiché la compressione funziona eliminando le ripetizioni, se l'attaccante indovina una parte del segreto (come un cookie di sessione), il pacchetto compresso sarà più piccolo. Procedendo per tentativi, è possibile recuperare l'intero segreto.

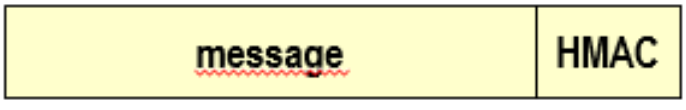
SECURITY FOCUS: Side-Channel Attack: CRIME

L'attacco **CRIME** sfrutta una proprietà fondamentale della compressione: i dati ripetuti occupano meno spazio. Immagina che il segreto sia "Cookie: SECRET". Un attaccante induce il browser a inviare "Cookie: SECRET" + "Cookie: GUESS". Se "GUESS" è sbagliato, la compressione è bassa. Se l'attaccante invia "Cookie: SECRET" + "Cookie: S", la "S" ripetuta viene compressa, rendendo il pacchetto totale più corto. Monitorando la lunghezza del pacchetto cifrato (canale laterale), l'attaccante può indovinare il segreto carattere per carattere, pur senza decifrare nulla.

Conclusione attuale: Oggi la compressione a livello TLS è considerata insicura e disabilitata in tutti i moderni client e server (ed è stata rimossa in TLS 1.3).

11 Message Authentication Code (MAC)

Nel contesto del Record Protocol, una volta frammentato (e opzionalmente compresso) il messaggio, il passo successivo fondamentale è garantire l'**integrità** e l'**autenticità** dei dati. Questo avviene tramite il calcolo del MAC.



11.1 Calcolo del MAC (MAC Computation)

Come illustrato nella slide, la computazione del MAC non avviene nel vuoto, ma dipende strettamente da parametri stabiliti in precedenza.

- **La Chiave Segreta (Simmetrica):** Il MAC richiede una chiave. Questa è una chiave *simmetrica* (condivisa tra client e server) che viene **derivata** dai parametri di sicurezza negoziati durante la fase di Handshake.
  - *Nota:* Non si tratta della chiave privata del certificato, ma di una "session key" specifica (spesso chiamata `write_MAC_key`) generata dinamicamente per quella specifica sessione.
- **La Funzione di Hash:** L'algoritmo matematico da utilizzare (es. SHA-1, SHA-256) viene definito durante l'**Handshake**. Client e Server si accordano su quale funzione usare all'interno della scelta della *Cipher Suite*.
- **La Costruzione HMAC (RFC 2104):** Il TLS utilizza specificamente lo standard **HMAC** (*Keyed-Hashing Message Authentication Code*).
  - Non basta fare un semplice Hash del messaggio concatenato alla chiave. HMAC è una costruzione nidificata specifica (definita nella RFC 2104) che è molto più robusta contro certi tipi di attacchi crittografici (come il *length extension attack*).

11.2 Struttura del Dato

Il risultato finale di questa operazione è visibile nello schema in basso alla slide:

$$\text{Output} = [\ \text{Message} \mid \text{HMAC}\ ]$$

Il tag HMAC viene appeso alla fine del messaggio (o del frammento compresso). Quando il destinatario riceve il pacchetto: 1. Separa il messaggio dal MAC. 2. Ricalcola il MAC sui dati ricevuti usando la propria copia della chiave simmetrica. 3. Confronta il MAC calcolato con quello ricevuto. Se coincidono, l'integrità è verificata.

12 Cifratura (Encryption)

Dopo aver frammentato il messaggio e calcolato il MAC, il protocollo procede alla cifratura per garantire la riservatezza (privacy) della comunicazione.

12.1 Oggetto della Cifratura

La slide chiarisce esattamente *cosa* viene cifrato. In linea con il paradigma "MAC-then-Encrypt" (visto nelle slide precedenti), l'operazione di cifratura si applica a:

- Il frammento di dati (eventualmente compresso).
- Il codice MAC associato.

Tutto ciò diventa un unico blocco illeggibile ("ciphertext").

Eccezioni (Quando non si cifra)

Esistono casi specifici in cui la cifratura non è possibile o non viene applicata:

- **Negoziazione Nulla:** Se le parti hanno negoziato una *Cipher Suite* con "Null Encryption" (usata solo per debug, poiché non offre privacy).
- **Fasi iniziali dell'Handshake:** I primissimi messaggi scambiati (come `ClientHello` e `ServerHello`) viaggiano necessariamente in chiaro, poiché le chiavi crittografiche non sono ancora state generate o scambiate.

12.2 Algoritmi a Chiave Simmetrica

La cifratura TLS utilizza algoritmi **simmetrici** (più veloci ed efficienti per grandi moli di dati rispetto alla crittografia asimmetrica).

Tipologie di Cifrari: L'algoritmo viene scelto durante la negoziazione iniziale. Può essere di due tipi:

- **Stream Cipher:** Cifrano il flusso bit per bit (es. RC4, oggi obsoleto).
- **Block Cipher:** Dividono i dati in blocchi di dimensione fissa (es. 3DES, AES).

Il Padding (per Block Cipher): Se si utilizza un cifrario a blocchi (come AES), la lunghezza totale dei dati deve essere un multiplo esatto della dimensione del blocco. Se i dati non raggiungono tale lunghezza, è necessario aggiungere del riempimento (**Padding**) per completare l'ultimo blocco.

Gestione delle Chiavi:

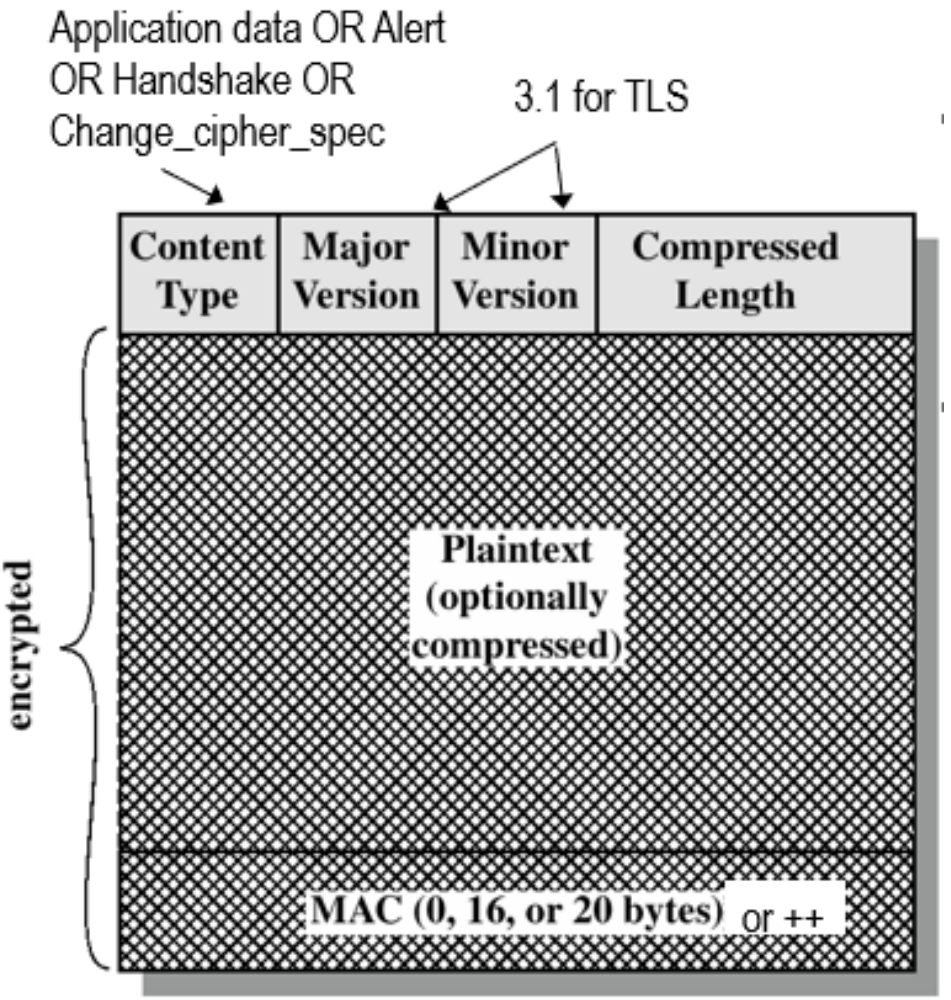
- La chiave segreta viene derivata dai parametri di sicurezza negoziati nell'Handshake.
- **Punto Critico:** La chiave usata per cifrare è **diversa** dalla chiave usata per calcolare il MAC. Anche se entrambe derivano dallo stesso "Master Secret", separare le chiavi per integrità e confidenzialità è una best practice di sicurezza fondamentale.

12.3 Vincolo sulle Dimensioni

La slide impone un limite tecnico all'overhead introdotto dalla cifratura: l'algoritmo di encryption non può aumentare la dimensione del messaggio originale di oltre **1024 bytes**. Questo margine serve a accomodare il padding e l'eventuale vettore di inizializzazione (IV), impedendo però un'espansione incontrollata dei dati.

13 Struttura del Pacchetto (Record Protocol Data Unit)

Il risultato finale di tutte le operazioni precedenti (frammentazione, compressione, MAC e cifratura) è la **Record Protocol Data Unit (PDU)**. Questa struttura è composta da un'intestazione in chiaro (Header) e un corpo cifrato.



13.1 L'Intestazione (Header)

Ogni record TLS inizia con un header di **5 byte** che viene trasmesso **in chiaro** (non cifrato), permettendo al ricevente di capire come interpretare il pacchetto in arrivo.

- **Content Type (1 byte):** Indica il tipo di protocollo di livello superiore contenuto nel payload (vedi sotto).
- **Version (2 bytes):** Indica la versione del protocollo.
  - Il primo byte è la *Major Version*, il secondo la *Minor Version*.
  - *Nota storica:* Nella slide viene citato "3.1 for TLS". Questo perché internamente TLS 1.0 è numerato come SSL 3.1 (SSL 3.0 era 3.0).
- **Length (2 bytes):** Indica la lunghezza del campo successivo (il frammento compresso e cifrato). Essendo di 2 byte, permette di gestire dimensioni fino a $2^{16} - 1$ (65535), sufficienti per il limite di frammentazione di 16KB.

13.2 I Tipi di Contenuto (Content Type)

Il campo *Content Type* agisce come un "multiplexer". Permette di mescolare diversi tipi di messaggi sulla stessa connessione TCP. I valori standard sono:

20 (0x14) - Change Cipher Spec: Segnala il passaggio alla nuova strategia di cifratura appena negoziata.

21 (0x15) - Alert: Messaggi di avviso o errori fatali (es. chiusura connessione).

22 (0x16) - Handshake: Tutti i messaggi relativi alla negoziazione della sicurezza (ClientHello, ServerHello, Certificate, ecc.).

23 (0x17) - Application Data: I dati veri e propri dell'applicazione (es. il contenuto della pagina HTTP).

13.3 Il Corpo del Pacchetto (Payload)

La parte sottostante l'header (rappresentata dalla trama incrociata nella slide) è il contenuto protetto.

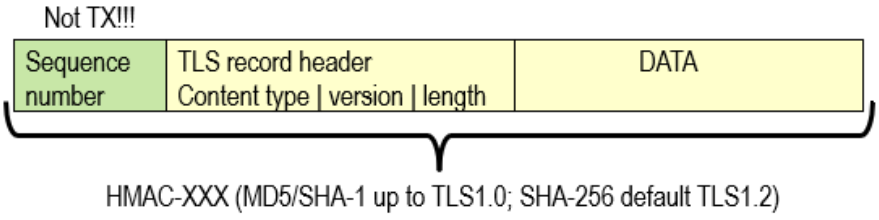
- **Cifratura:** Tutto ciò che si trova qui è cifrato (a meno che non siamo nella fase iniziale dell'handshake o si usi una cifratura NULL).
- **Contenuto:** Include:
  1. Il **Plaintext** (il frammento di dati originale, opzionalmente compresso).
  2. Il **MAC** (appendice di integrità), che può variare tipicamente da 0, 16 a 20 byte (o più, a seconda dell'algoritmo di hash scelto, es. MD5 vs SHA-1).

14 Prevenzione del Replay: I Numeri di Sequenza

Per mitigare il rischio di Replay Attack (l'invio ripetuto di pacchetti validi intercettati) e garantire l'ordine corretto dei dati, il TLS introduce il concetto di **Numeri di Sequenza** (Sequence Numbers).

14.1 Inclusione nel MAC (MAC Generation Details)

La soluzione adottata dal TLS è elegante ed efficiente. Invece di inviare il numero di sequenza come campo extra in ogni pacchetto (che consumerebbe larghezza di banda), il numero viene reso **implicito**.



Come mostrato nello schema della slide:

- **Not Transmitted (Not TX!!!):** Il numero di sequenza non viaggia sulla rete. Non lo troverai nel pacchetto fisico.

- **Calcolo del MAC:** Tuttavia, il numero di sequenza viene utilizzato come **input matematico** per il calcolo del MAC.

$$\text{MAC} = \text{HMAC}(\text{Key}, \textbf{SeqNum} + \text{Header} + \text{Data})$$

- **Verifica:** Quando il server riceve un pacchetto, calcola il MAC usando il "suo" contatore locale del numero di sequenza atteso.
  - Se il pacchetto è quello giusto, i MAC coincidono.
  - Se è un pacchetto vecchio (Replay), il server userà un numero di sequenza diverso (attuale) rispetto a quello usato per generare il MAC del pacchetto (vecchio). Il controllo fallirà e il pacchetto verrà scartato.

SECURITY FOCUS: Attacco di Replay (Replay Attack)

Un **Replay Attack** avviene quando un attaccante intercetta un pacchetto cifrato valido (es. "Trasferisci 100 euro") e lo reinvia identico al server in un secondo momento. Senza numeri di sequenza, il server accetterebbe il pacchetto come autentico ed eseguirebbe l'operazione una seconda volta. Rendendo il **SeqNum** parte del calcolo del MAC, ogni pacchetto diventa matematicamente unico nel tempo: reinviare lo stesso pacchetto in futuro causerà un errore di verifica del MAC, poiché il SeqNum del server sarà avanzato.

14.2 Gestione dei Sequence Numbers

I numeri di sequenza sono gestiti come stati interni alle due estremità della connessione (Client e Server).

Stato locale: Sia il Client che il Server mantengono dei contatori:

- `C_seqnum`: Il contatore del Client.
- `S_seqnum`: Il contatore del Server.

Distinti per direzione: Esistono contatori separati per il traffico in lettura e in scrittura.

Dimensione e Limiti:
- Sono interi a **64 bit**.
- Inizializzati a 0 all'inizio della sessione.
- Possono arrivare fino a $2^{64} - 1$.
- **Regola fondamentale:** Non devono **mai ricominciare da capo** (Do NOT wrap). Se si raggiunge il limite (evento rarissimo data la grandezza del numero), è obbligatorio rinegoziare le chiavi.

14.3 Implicazioni sul Livello di Trasporto (TCP vs DTLS)

La scelta di *non trasmettere* il numero di sequenza ha una conseguenza architetturale profonda:

- **Dipendenza da TCP:** Poiché il numero è implicito, il TLS assume che non ci siano "buchi" nella sequenza dei pacchetti. Si affida totalmente all'affidabilità del livello di trasporto sottostante. Ecco perché TLS richiede un protocollo affidabile come TCP (che garantisce l'ordine e la riconsegna).
- **Il caso DTLS (Datagram TLS):** Se si usa UDP (che non è affidabile e perde pacchetti), questa logica si rompe.
  - *Soluzione:* In DTLS, non potendo contare sull'ordine implicito, il numero di sequenza deve essere aggiunto **esplicitamente** nel pacchetto.
  - DTLS aggiunge quindi 8 byte (2 byte di epoch + 6 byte di sequence number) ad ogni record per gestire la perdita o il disordine dei pacchetti tipico dell'UDP.