TLS 1.3 e Perfect Forward Secrecy

Christian Sfeir

December 5, 2025

1 TLS 1.3: Introduzione

Il **TLS 1.3 (Transport Layer Security)** rappresenta l'evoluzione più significativa del protocollo di sicurezza standard per le comunicazioni via Internet. Oltre ai miglioramenti in termini di performance (velocità di connessione), il focus principale di questa versione risiede nel rafforzamento drastico della sicurezza crittografica.

Il pilastro fondamentale di questa architettura è la **Perfect Forward Secrecy (PFS)**, o "Segretezza in Avanti Perfetta".

La PFS è una proprietà dei protocolli di negoziazione delle chiavi che garantisce la sicurezza delle sessioni passate anche in caso di compromissione futura delle chiavi a lungo termine.

- **Il problema (Senza PFS)** è che in passato, se un attaccante registrava il traffico cifrato di una conversazione e, mesi o anni dopo, riusciva a rubare la chiave privata del server (chiave a lungo termine), poteva utilizzare quella chiave per decifrare retroattivamente tutto il traffico registrato.
- **La soluzione (con PFS in TLS 1.3)** è che il protocollo obbliga l'uso di chiavi di sessione **effimere** (usa e getta). Inoltre, per ogni singola connessione viene generata una coppia di chiavi unica che viene distrutta al termine della sessione. Anche se la chiave del server viene comporomessa in futuro, l'attaccante non può decifrare le conversazioni passate perché le chiavi specifiche di quelle sessioni non esistono più e non possono essere derivate dalla chiave del server.

2 Il processo di Approvazione (Marzo 2018)

La definizione di un protocollo così critico non è immediata, ma segue un rigoroso iter burocratico e tecnico all'interno della **IETF (Internet Engineering Task Force)**. Il momento formale che ha sancito la maturità tecnica del protocollo è avvenuto il **21 Marzo 2018**. In questa data, il documento tecnico è passato allo stato di "Proposed Standard". Questo status indica che la comunità tecnica ha raggiunto un consenso sul design del protocollo e lo ritiene pronto per l'implementazione, sebbene possa ancora subire affinamenti minori. Il documento approvato portava il nome tecnico **draft-ietf-tls-tls13-28**. Il numero "28"è significativo: indica che il protocollo ha subito ben **28 revisioni (bozze)** prima di essere ritenuto abbastanza sicuro e stabile per l'approvazione.

Questo sottolinea l'intenso lavoro di revisione e correzione delle vulnerabilità effettuato dagli ingegneri.

3 La Pubblicazione Ufficiale: RFC 8446 (Agosto 2018)

Dopo l'approvazione di marzo, il protocollo è stato definitivamente pubblicato e indicizzato nell'agosto del 2018. Il documento finale è indetificato come **RFC 8446**. Un RFC (Reuqest for Comments) è il documento ufficiale che definisce gli standard di Internet. L'RFC 8446 definisce le specifiche finali di TLS 1.3.

- **Punti salienti del documento:**
  - **Obsolescenza dei vecchi standard:** L'RFC 8446 dichiara esplicitamente che **TLS 1.3 rende obsoleti**(sostituisce) i precedenti standard. Nello specifico, viene reso obsoleto l'**RFC 5246**, che definiva il **TLS 1.2**. Vengono anche rimossi meccanismi precedenti come quelli definiti negli RFC 5077 e 6961. Questo segna un taglio netto con il passato per eliminaree configurazioni insicure.
  - **Obiettivo di Sicurezza Dichiarati:** Il documento specifica che lo scopo del protocollo è permettere comunicazioni client/server prevedendo tre tipologie di attacco:
    - **Eavesdropping (intercettazione):** la capacità di terzi di ascoltare e leggere i dati riservati.
    - **Tampering (Manomissione):** la capacità di terzi di modificare i dati mentre viaggiano dal mittente al destinatario senza che questi se ne accorgano.
    - **Message Forgery (Falsificazione):** la capacità di terzi di iniettare messaggi falsi fingendosi un'entità fidata.
  - Il documento è stato curato principalmente da E.Rescorla (Mozilla), riflettendo la stretta collaborazione tra gli organismi di standardizzazione e i produttori di browser per garantire la sicurezza del web.

4 TLS 1.3: weak ciphers' pruning

In TLS 1.3 si è deciso di eliminare tutto ciò che è debole o "legacy". Hanno rimosso tutti gli algoritmi simmetrici considerati obsoleti:

- **Gli hash insicuri: SHA-1 e MD5**. Ormai si possono trovare "collisioni" (due file diversi con lo stesso hash) troppo facilmente, quindi non ci si può più fidare.
- **I vecchi cifrari:** Hanno rimosso **RC4,DES e 3DES**. Questi erano i pilastri degli anni '90, ma oggi si rompono troppo facilmente con la potenza di calcolo moderna.
- **La modalità AES-CBC:** AES in sé è ancora ottimo, ma la modalità CBC (Cipher Block Chaining) è stata eliminata. Questo perché era molto difficile da implementare senza creare buchi di sicurezza (i famosi attacchi sul "Padding".

Cosa è rimasto? La regola principale di TLS 1.3 è semplice: Tutto ciò che è rimasto DEVE essere **AEAD**.

- **AEAD** sta per *Authenticated Encryption with Associated Data*. In poche parole, questi algoritmi fanno due cose insieme in un colpo solo:
  - Cifrano i dati (privacy).
  - Verificano che non siano stati toccati (integrità).
- Il vantaggio enorme è che usando solo AEAD, si è riusciti a eliminare una clsse intera di attacchi chiamati **CCA (Chosen Ciphertext Attacks)** come "Padding Oracle" o "Lucky 13" (che sfruttavano le debolezze di come i vecchi sistemi gestivano gli errori di decifrazione) non possono più funzionare in questo contesto.

Anche i nomi delle "Cipher Suites" sono cambiati. Ora sono molto più corti e seguono questo schema: **TLS_AEAD_HASH**. Significa che la suite ci dice solo quale algoritmo di cifratura (AEAD) viene usato e quale funzione di Hash viene usata per derivare le chiavi (HKDF):

- **TLS_AES_128_GCM_SHA256:** veloce e sicuro.
- **TLS_AES_256_GCM_SHA384:** veloce e sicuro, ma con chiavi più lunghe.
- **TLS_CHACHA20_POLY1305_SHA256:** L'alternativa moderna. E' ottima sugli smartphone perché è velocissima anche senza chip dedicati (a differenza di AES).
- **TLS_AES_128_CCM_SHA256:** Una variante usate spesso nell'IoT.
- **TLS_AES_128_CCM_8_SHA256:** Simile alla precedente, ma con un tag di autenticazione più corto (8 byte).

In sintesi, si è tolta la complessità e le opzioni pericole. Ora c'è meno scelta, ma è quasi impossibile scegliere una configurazione insicura.

5 TLS 1.3 Handshake

Fino alla versione 1.2, il protocollo supportava ben 4 metodi diversi per scambiarsi le chiavi pubbliche:

- **RSA Key Transport:** era il metodo più comune. Semplice, ma con un grosso difetto: non garantiva la sicurezza futura (niente Forward Secrecy).
- **Anonymous Diffie-Hellman:** Scambio di chiavi senza autenticazione. Pericolodo perché chiunque poteva mettersi in mezzo (Man-in-the-middle).
- **Fixed Diffie-Hellman:** usava parametri statici, fissi.
- **Diffie-Hellman Ephemeral (DHE):** L'unico metodo che generava chiavi temporanee per ogni sessione.

Con TLS 1.3 hanno rimosso tutti i metodi della lista sopra, tranne uno (praticamente hanno preso un machete e hanno tagliato tutto dio porco!):

- **Diffie-Hellman Ephemeral**

Perché questa scelta drastica? Ci sono due motivi principali:

- **Forward Secrecy:** Obbligando l'uso di chiavi effimere, si garantisce che se un hacker ruba la chiave privata del server tra un anno, non potrà decifrare le conversazioni registrate oggi.
- **Problemi di sicurezza di RSA:** il vecchio metodo RSA aveva delle vulnerabilità note (come il Bleichenbacker Oracle,...). Rimuovendo il supporto RSA per lo scambio chiavi, si è eliminato il problema alla base.

6 Che co'è la Perfect Forward Secrecy (PFS)?

E' una proprietà dei protocolli crittografici che protegge le sessioni di comunicazione passate anche se le chiavi a lungo termine vengono compromesse in futuro.
Requisito minino: Il primo livello di sicurezza che ci si aspetta è che la compromissione di una singola chiave di sessione sia un evento isolato.

- **Isolamento della sessione:** Se un attaccante riesce in qualche modo a ottenere la chiave simmetrica usata per cifrare una specifica sessione, il danno deve essere limitato a quella sola sessione.
- **Protezione di passato e futuro:** questa compromissione non deve permettere all'attaccante di decifrare i dati scambiati in sessioni precedenti né quelli che verranno scambiati in sessioni future. Ogni sessione deve essere una cosa a sé stante.

La garanzia reale (Actuallt more than this) La PFS va oltre il requisito minimo ed è qui che risiede la sua vera potenza. Affronta lo scenario peggiore: la compromissione della "chiave maestra".

- **Lo scenario di attacco:** Immaginiamo che un attaccante abbia registrato per mesi tutto il traffico cifrato tra te e un server. Non può leggerlo perché è cifrato.
- **Il furto della chiave privata:** Oggi, l'attaccante riesce a hackerare il server e a rubare la sua chiave privata a lungo termine.
- **La protezione PFS:**
  - **Senza PFS (es. vecchio scambio RSA):** L'attaccante potrebbe usare la chiave privata rubata oggi per decifrare retroattivamente tutto il traffico registrato nei mesi scorsi.
  - **Con PFS:** La compromissione della chiave privata a lungo termine non influisce sui dati consegnati prima di quel momento. L'attaccante non può usare la chiave privata di oggi per ricalcolare la chiavi di sessione effimere usate ieri, perché quelle chiavi sono state distrutte subito dopo l'uso e non sono mai state cifrate direttamente con la chiave a lungo termine.

L'essenza della PFS La PFS garantisce che le chiavi di sessione (quelle che cifrano i dati veri e propri) non vengano compromesse nemmeno se i segreti a lungo termine (come la chiave private del server) usati durante la fase iniziale di scambio delle chiavi vengono compromessi.

7 L'importanza critica della Perfect Forward Secrecy

L'adozione della PFS è divenuta indispensabile per difendersi contro un **modello di attaccante più potente** rispetto a quello considerato in passato. Questo nuovo modello di minaccia si basa sulla combinazione di due capacità offensive specifiche che, se unite, sarebbero catastrofiche senza la protezione della PFS.

1.Capacità di Archiviazione Massiva ("Logging huge amount of data"). Il primo pilastro del nuovo modello di minaccia è la capacità dell'avversario di registrare e conservare indefinitamente enormi volummi di traffico cifrato. L'attaccante adotta la strategia del "store now, decrypt later" (memorizza ora, decifra dopo). Anche se al momento non possiede la chiave per leggere i dati, li archivia in attesa di ottenere la chiave in futuro.

2.Capacità di compromissione occasionale delle chiavi ("occasionally break a key pair"). Il secondo pilastro riguarda la realistica probabilità che una coppia di chiavi pubblica/privata venga compromessa nel corso del tempo. Non si assume necessariamente che l'attaccante rompa l'algoritmo crittografico (es.RSA) matematicamente, ma che sfrutti **bug software nell'implementazione**.

E' qui che i due punti si uniscono. Se un attaccante ha archiviato anni di traffico (punto 1) e poi sfrutta un bug per rubare la chiave (punto 2), senza PFS potrebbe decifrare retroattivamente tutto l'archivio.

8 Il meccanismo del "RSA Key Transport": no PFS!

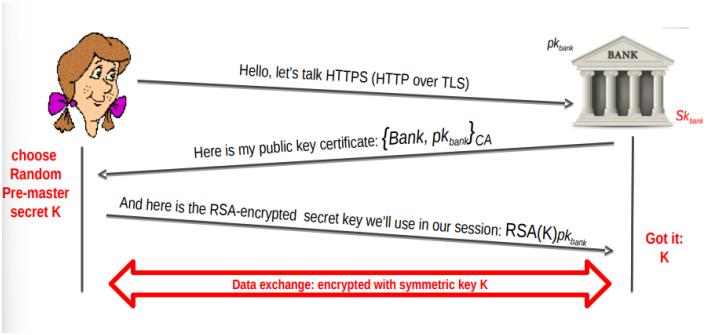

Figure 1: RSA Key Transport

La *Figura 1* illustra il funzionamento del handshake tradizionale basato su RSA, evidenziando il difetto strutturale che impedisce la PFS.

- **1:** Il client inizia la connessione ("Hello") e il server risponde inviando il proprio certificato digitale contenente la sua **chiave pubblica** (pk_{bank}).
- **2:** Il client genera in locale una chiave casuale (il "Pre-master secret" K).
- **3:** Il client cifra questa chiave K utilizzando la chiave pubblica del server e invia il crittogramma risultante ($RSA(K)pk_{bank}$) attraverso la rete.
- **4:** il server riceve il pacchetto e utilizza la propria **chiave privata** (Sk_{bank}) per decifrare il messaggio e ottenere K.

Il punto critico: la segretezza della chiave di sessione K dipende interamente ed esclusivamente dalla riservatezza della chiave privata del server Sk_{bank}. Il segreto viene trasmesso (anche se cifrato), creando un legame persistente tra la chiave a lungo termine e ogni singola sessione.

Ora espandiamo il concetto precedente mostrando una timeline che concretizza l'attacco di tipo "Store Now, Decrypt Later".

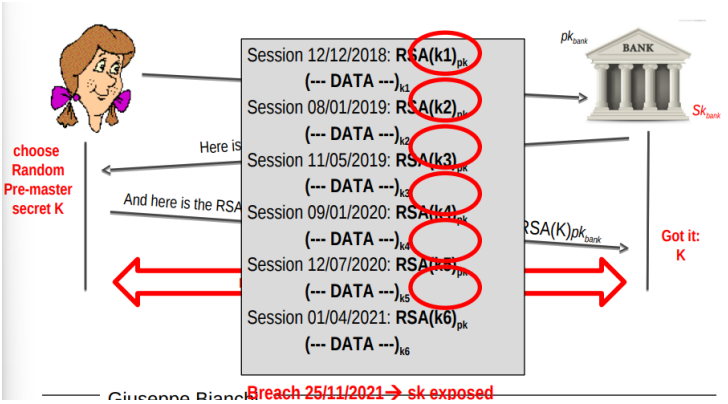

Figure 2: RSA Key Transport

In *Figura 2* viene mostrato un arco temporale che va dal 2018 al 2021. In questo periodo avvengono diverse sessioni, ognuna protetta da una chiave diverse $(k1, k2, ..., k6)$. Un attaccante passivo registra tutto il traffico di rete. Ogni sessione registrata contiene la chiave cifrata: $RSA(kn)_{pk}$. Il 25/11/2021 avviene una violazione di sicurezza e la chiave privata del server (sk_{bank}) viene esposta. Poiché l'attaccante possiede ora la sk_{bank}, può applicarla matematicamente a tutti i messaggi $RSA(kn)$ registrati anni prima. Il risultato è che l'attaccante ottiene in chiaro tutte le chiavi passate $(k1, k2, ..., k6)$ e può decifrare tutti i dati storici associati. La compromissione avvenuta nel 2021 distrugge la sicurezza delle sessioni del 2018.

9 La vulnerabilità del "Fixed Diffie-Hellman

Spesso si assume erroneamente che l'uso dell'algoritmo Diffie-Hellman (DH) garantisca automaticamente la PFS. In *Figura 3* si dimostra che non è così se i parametri sono statici.

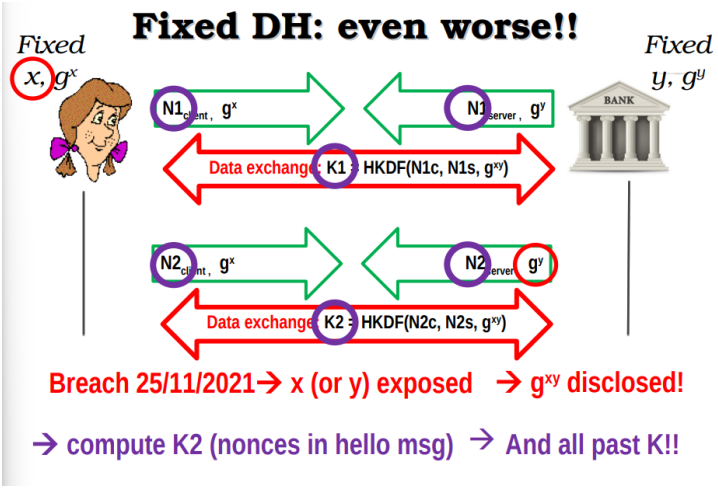

Figure 3: Fixed DH

- **Configurazione Statica:** Nello schema "Fixed DH", il server (e talvolta il client) riutilizza gli stessi parametri segreti (y) e pubblici (g^y) per più sessioni, invece di rigenerarli ogni volta.
- **Derivazione della chiave:** la chiave di sessione ($K1, K2$) viene calcolata tramite una funzione (HKDF) che combina dei valori casuali (nonces $N1, N2$) con il segreto condiviso calcolato matematicamente (g^{xy}).
- **Il fallimento della sicurezza:** Se il 25/11/2021 viene esposto l'esponente segreto statico del server (y):
  - L'attaccante ha già registrato i valori pubblici g^x scambiati in passato.
  - Conoscendo y e g^x, l'attaccante può calcolare il valore g^{xy}.
  - Poiché g^{xy} è la radice di sicurezza ed è costante per tutte le sessioni (essendo x e y fissi), la sua compromissione permette di ricalcolare tutte le chiavi di sessione passate ($K1, K2, ...$), nonostante l'uso dei nonces.

Quindi, in conclusione, l'uso di parametri statici rende Diffie-Hellman insicuro quanto RSA rispetto alla PFS. Solo la variante **Ephemeral (DHE)**, che cambia x e y a ogni connessione, garantisce PFS.

10 Lo Standard Sicuro: Diffie-Hellman Ephemeral (DHE)

Questa sezione analizza l'architettura crittografica adottata da TLS 1.3 per garantire la **Perfect Forward Secrecy (PFS)**. Il protocollo abbandona i metodi di scambio chiavi statici (come RSA Key Transport) in favore di uno scambio basato su parametri effimeri e autenticati.

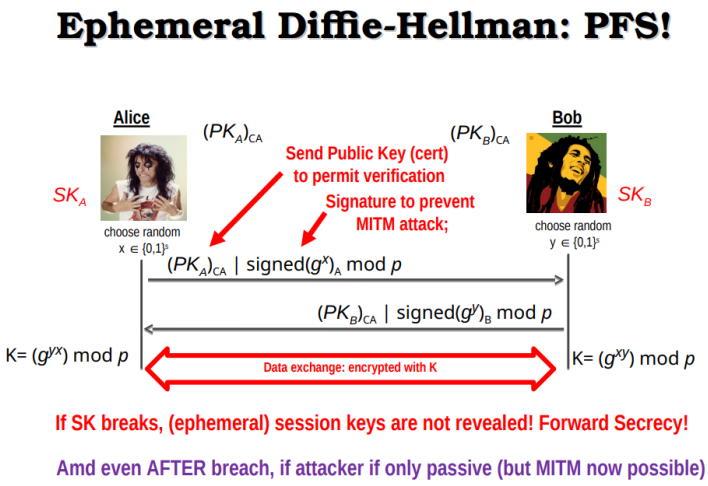

Figure 4: Ephemeral Diffie-Hellman: PFS

10.1 I Protagonisti e il Ruolo dell'Autenticazione

A differenza dello scambio anonimo (vulnerabile), in TLS 1.3 l'identità delle parti viene rigorosamente verificata per prevenire attacchi di tipo *Man-in-the-Middle* (MITM).

- **Chiavi a Lungo Termine:** Alice (client) e Bob (server) possiedono ciascuno una coppia di chiavi stabili nel tempo:
  - Una chiave privata segreta (SK_A per Alice, SK_B per Bob).
  - Una chiave pubblica distribuita tramite certificato digitale firmato da una CA ($(PK_A)_{CA}$, $(PK_B)_{CA}$).
- **Utilizzo delle Chiavi:** È fondamentale notare che queste chiavi a lungo termine **NON** vengono utilizzate per cifrare il traffico o per scambiare direttamente il segreto. Il loro unico scopo è **firmare digitalmente** i parametri dello scambio per garantirne l'autenticità.

10.2 Il Flusso dello Scambio Effimero (The Ephemeral Exchange)

La sicurezza del protocollo risiede nella natura temporanea ("usa e getta") dei parametri matematici utilizzati. Il processo avviene in tre fasi:

10.2.1 1. Generazione dei Parametri

Per ogni singola sessione, le parti generano nuovi parametri casuali:

- **Alice** sceglie un numero casuale segreto $x \in \{0,1\}^s$ e calcola il valore pubblico $g^x \pmod p$.
- **Bob** sceglie un numero casuale segreto $y \in \{0,1\}^s$ e calcola il valore pubblico $g^y \pmod p$.

Nota: x e y sono effimeri, ovvero vengono rigenerati ad ogni connessione.

10.2.2 2. Scambio e Firma (Protezione MITM)

Per evitare che un attaccante attivo intercetti e modifichi i valori pubblici (g^x o g^y), questi vengono firmati:

- Alice invia: Il proprio certificato $(PK_A)_{CA}$ e il valore g^x accompagnato dalla firma digitale calcolata con SK_A: signed$(g^x)_A$.
- Bob invia: Il proprio certificato $(PK_B)_{CA}$ e il valore g^y accompagnato dalla firma digitale calcolata con SK_B: signed$(g^y)_B$.

10.2.3 3. Calcolo della Chiave di Sessione

Dopo aver verificato le firme (e quindi l'identità dell'interlocutore), entrambe le parti calcolano la chiave comune K:

$$K = (g^y)^x \pmod p \quad \text{(Calcolato da Alice)}$$

$$K = (g^x)^y \pmod p \quad \text{(Calcolato da Bob)}$$

Il risultato è identico matematicamente (g^{xy}), ma il segreto finale non ha mai attraversato la rete.

10.3 La Garanzia di Perfect Forward Secrecy (PFS)

L'architettura DHE offre una protezione robusta contro la compromissione futura delle chiavi a lungo termine.

1. **Scenario di Compromissione:** Supponiamo che un attaccante riesca a rubare la chiave privata a lungo termine di Bob (SK_B) in un momento futuro.
2. **Impatto Limitato:**
   - L'attaccante potrà effettuare attacchi MITM su sessioni *future* (falsificando la firma di Bob).
   - L'attaccante **NON potrà decifrare le sessioni passate** registrate ("Forward Secrecy").
3. **Il Motivo Tecnico:** Per ricalcolare la chiave di una vecchia sessione ($K = g^{xy}$), l'attaccante avrebbe bisogno dei valori segreti effimeri x o y. Tuttavia, poiché tali valori erano effimeri, sono stati cancellati dalla memoria di Alice e Bob immediatamente dopo la fine della sessione. La chiave SK_B rubata serviva solo a firmare, non a generare la chiave di cifratura, rendendo il traffico storico matematicamente inaccessibile.

11 Perfect Forward Secrecy vs Pre-shared Key

Questa sezione risponde a una domanda cruciale: **È possibile avere la sicurezza futura (PFS) se usiamo una chiave statica fissa (come una password del Wi-Fi)?**

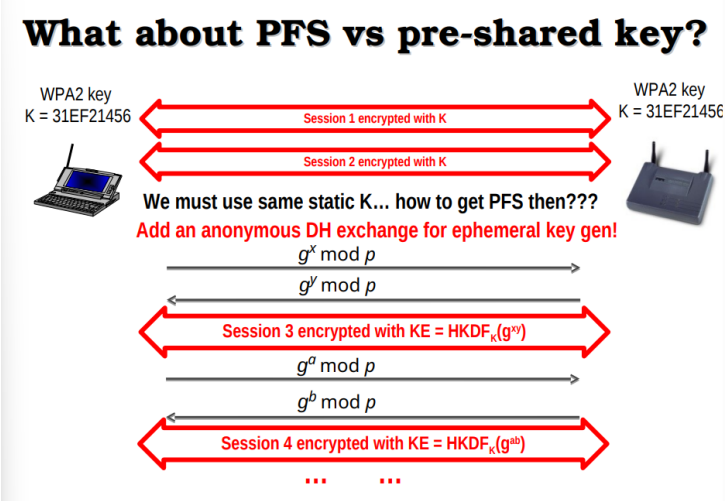

Figure 5: PFS vs Pre-shared Key

11.1 Lo Scenario Tradizionale (e il suo problema)

Nella parte alta dell'immagine viene mostrato lo scenario classico, simile a quello del protocollo WPA2 (quello che usiamo comunemente per il Wi-Fi domestico).

- **La Chiave Statica:** Abbiamo un laptop e un router (Access Point). Entrambi conoscono una chiave segreta a lungo termine, chiamata PSK (Pre-Shared Key) o WPA2 key. Nell'esempio, la chiave è $K = 31EF21456$.
- **Il Funzionamento:**
  - Per avviare la Sessione 1, i dispositivi usano la chiave statica K per cifrare i dati.
  - Per avviare la Sessione 2, usano ancora la stessa chiave K (o chiavi derivate solo da essa).
- **Il Problema di Sicurezza:**
  - In questo scenario **NON c'è PFS**.
  - Poiché la chiave K è statica e non cambia mai, se un attaccante riesce a scoprirla (magari tra un anno), potrà decifrare **tutte** le sessioni passate (Sessione 1, Sessione 2, ecc.) che ha registrato. La sicurezza dell'intero storico dipende da un'unica password statica.

11.2 La Soluzione: Combinare PSK e Diffie-Hellman

La parte centrale dell'immagine si introduce la soluzione tecnica per introdurre la PFS anche in sistemi basati su password. L'idea è: *"Aggiungere uno scambio Diffie-Hellman anonimo per generare una chiave effimera"*.

Ecco come funziona il nuovo protocollo ibrido:

A. Lo Scambio Effimero (Anonymous DH)

Invece di usare subito la chiave statica K per cifrare, i due dispositivi eseguono prima uno scambio matematico:

- Il client invia una parte pubblica casuale: $g^x \bmod p$.
- Il server risponde con la sua parte pubblica casuale: $g^y \bmod p$.
- Entrambi calcolano il segreto condiviso matematico g^{xy}, che è temporaneo e legato solo a questa sessione.

Nota: Questo scambio è detto "anonimo" perché non usiamo certificati digitali per autenticare g^x e g^y. L'autenticazione avverrà nel passaggio successivo.

B. La "Ricetta" per la Chiave di Sessione (HKDF)

Per creare la chiave di cifratura vera e propria (KE), usiamo una funzione speciale chiamata HKDF (HMAC-based Key Derivation Function). La formula mostrata nella slide è:

$$\text{KE} = \text{HKDF}(K, g^{xy})$$

Cosa significa? Significa che la chiave di sessione KE viene creata mescolando due ingredienti fondamentali:

- **L'ingrediente Statico (K):** La chiave Pre-Shared Key (31EF...). Questo serve per l'autenticazione. Solo chi conosce la password corretta può generare la chiave finale corretta.
- **L'ingrediente Effimero (g^{xy}):** Il risultato dello scambio Diffie-Hellman. Questo serve per la Forward Secrecy. Cambia ad ogni connessione.

11.3 Il Risultato: Sicurezza Totale

Guardando la parte bassa dell'immagine (Sessione 3 e Sessione 4), vediamo il risultato pratico:

- **Sessione 3:** Usa una chiave KE derivata dalla password K + lo scambio g^{xy}.
- **Sessione 4:** I dispositivi fanno un nuovo scambio Diffie-Hellman (con nuovi numeri a e b), ottenendo g^{ab}. La nuova chiave di sessione sarà derivata dalla password K + il nuovo g^{ab}.

Perché questo garantisce la PFS? Se un hacker domani ruba la tua password statica K (la WPA2 key):

- Potrà collegarsi alla rete in futuro (rompendo l'autenticazione).
- **MA NON potrà decifrare le sessioni 3 e 4 registrate nel passato.**
  - Perché? Perché per ricalcolare le chiavi di quelle sessioni (KE), l'hacker avrebbe bisogno non solo di K, ma anche dei valori effimeri x, y (o a, b) che sono stati cancellati dalla memoria dei dispositivi subito dopo l'uso.
  - Possedere solo la password statica non basta più per aprire la "cassaforte" del passato.

12 Forward Secrecy nelle App di Messaggistica

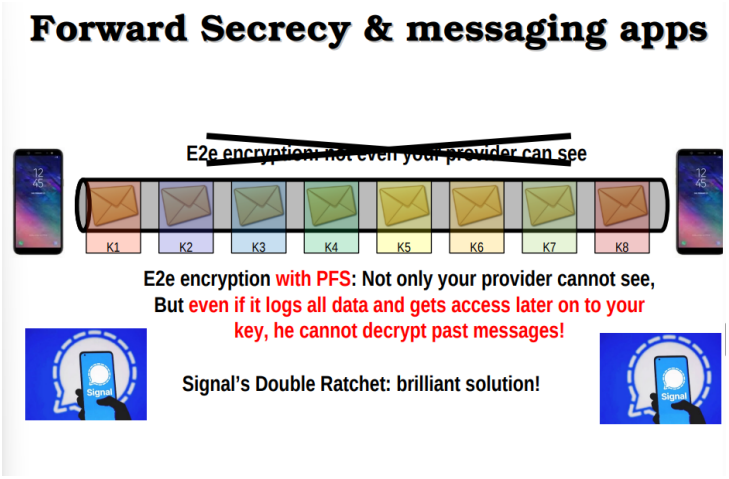

Figure 6: Forward Secrecy and messagging apps

Nella parte alta dell'immagine vediamo una frase barrata: "E2e encryption: not even your provider can see". Perché è cancellata? Non perché sia falsa, ma perché è incompleta.

- **La visione classica:** La crittografia End-to-End (E2E) garantisce che i messaggi vengano cifrati sul telefono del mittente e decifrati solo su quello del destinatario. Il provider (es. il server di WhatsApp o l'operatore telefonico) vede solo dati incomprensibili.
- **Il problema nascosto:** Se la crittografia E2E si basa su una chiave statica (che non cambia mai), siamo vulnerabili all'attacco "Store Now, Decrypt Later". Se il provider registra i messaggi cifrati oggi e tra un anno riesce a rubare la tua chiave privata, potrà leggere tutto lo storico.

12.1 La vera sicurezza: E2E Encryption CON PFS

L'immagine introduce il concetto fondamentale evidenziato in rosso: "E2e encryption with PFS". La Perfect Forward Secrecy aggiunge un livello di protezione temporale cruciale. Ecco le due garanzie che offre:

- **Privacy Presente:** Il provider non può leggere i messaggi mentre passano (questo lo faceva già la E2E base).
- **Privacy Passata (La vera novità):** La slide afferma: "even if it logs all data and gets access later on to your key, he cannot decrypt past messages!".
  - **Traduzione:** Anche se qualcuno (un hacker, il governo, o il provider stesso) registra tutto il traffico di rete per mesi e successivamente riesce a rubare la tua chiave attuale, non potrà decifrare i messaggi inviati in passato.
  - I messaggi vecchi (rappresentati dalle buste K1, K2, K3 nell'immagine) rimangono sigillati per sempre perché le chiavi usate per cifrarli sono state distrutte subito dopo l'uso.

12.2 La Soluzione Tecnica: Il "Double Ratchet" di Signal

Come si ottiene questa sicurezza nelle app reali? L'immagine cita esplicitamente l'app Signal e la sua soluzione definita "brilliant".

- **L'algoritmo Double Ratchet:** È il protocollo crittografico (usato oggi anche da WhatsApp) che gestisce le chiavi.
- **Come funziona (concettualmente):** Immagina un meccanismo a cricchetto (ratchet) che gira solo in avanti.
  - Per ogni singolo messaggio inviato, le chiavi vengono "aggiornate" o fatte ruotare.
  - La chiave del messaggio 5 (K5) viene usata per derivare la chiave del messaggio 6 (K6), e poi K5 viene distrutta immediatamente.
  - Se un attaccante ruba il telefono e trova la chiave K8, non può tornare indietro matematicamente per calcolare K7, K6 o K1.

In sintesi

Nelle moderne app di messaggistica la sicurezza non è statica. Grazie alla PFS e all'algoritmo Double Ratchet, una compromissione del dispositivo oggi non mette a rischio i segreti che ci siamo scambiati ieri.

13 L'Algoritmo Double Ratchet: Architettura e Funzionamento

L'algoritmo "Double Ratchet" rappresenta lo stato principale per la crittografia asincrona dei messaggi. Il nome deriva dalla combinazione di due meccanismi distinti ("ingranaggi" o "cricchetti") che lavorano in sinergia: il *Ratchet a Chiave Simmetrica* e il *Ratchet Diffie-Hellman*.

13.1 Il Ratchet a Chiave Simmetrica (KDF Chain)

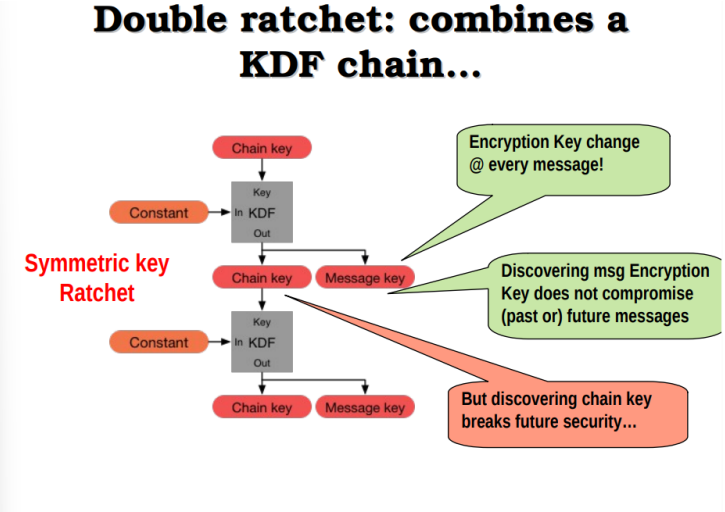

Figure 7: Il Ratchet a Chiave Simmetrica

Questa componente gestisce la sicurezza a livello di singolo messaggio. Il meccanismo si basa su una KDF (Key Derivation Function), una funzione crittografica che deriva nuove chiavi partendo da un segreto iniziale.

Funzionamento del Processo a Catena

Il diagramma mostra una struttura a catena ("chain") che avanza ad ogni messaggio inviato o ricevuto:

- **Input della KDF:** La funzione prende in ingresso due elementi:
  - Una *Chain Key* (chiave di catena) corrente.
  - Una costante predefinita.
- **Output della KDF:** La funzione produce due output distinti:
  - **Message Key (Chiave del Messaggio):** Questa è la chiave effimera utilizzata esclusivamente per cifrare il contenuto di quel singolo messaggio.
  - **Nuova Chain Key:** Questa chiave viene utilizzata come input per il passo successivo della catena.

Proprietà di Sicurezza del Ratchet Simmetrico

- **Forward Secrecy per messaggio:** L'immagine evidenzia in verde che la chiave di cifratura cambia a ogni messaggio ("Encryption Key change @ every message"). Poiché la *Message Key* viene derivata e poi immediatamente distrutta dopo l'uso, la compromissione di una singola chiave di messaggio non compromette i messaggi passati né quelli futuri ("Discovering msg Encryption Key does not compromise (past or) future messages").
- **Il Limite (Vulnerabilità):** L'immagine segnala un punto critico in rosso. Se un attaccante riesce a compromettere la *Chain Key*, la sicurezza futura è infranta ("But discovering chain key breaks future security..."). Dato che ogni chiave di catena deriva la successiva in modo deterministico, l'attaccante potrebbe calcolare tutte le chiavi future. È qui che diventa necessario il secondo ratchet.

13.2 Il Ratchet Diffie-Hellman (DH Ratchet)

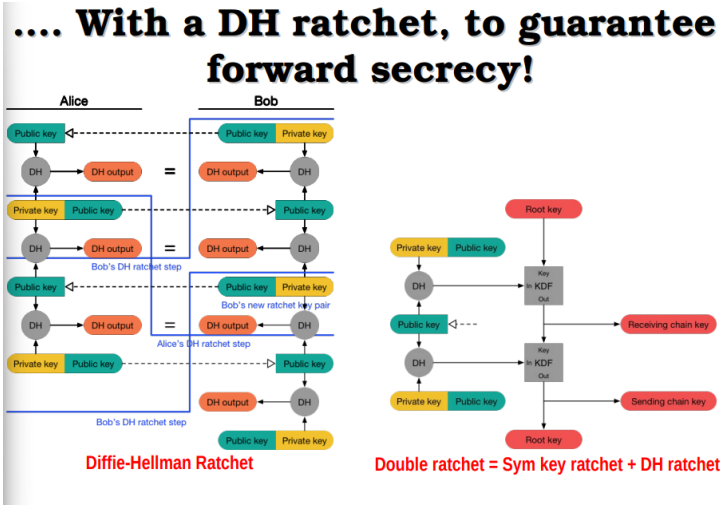

Figure 8: DH Ratchet

Per mitigare la vulnerabilità del solo ratchet simmetrico (ovvero la compromissione della catena), viene introdotto un secondo livello basato sulla crittografia asimmetrica: il Ratchet Diffie-Hellman. Questo meccanismo introduce nuova entropia nel sistema.

Funzionamento dello Scambio (Lato Sinistro dell'immagine)

Il diagramma mostra l'interazione tra Alice e Bob:

- **Scambio Continuo di Chiavi:** Alice e Bob non usano parametri fissi. Oltre ai messaggi, si scambiano nuove chiavi pubbliche Diffie-Hellman all'interno della conversazione.
- **Aggiornamento del Segreto:**
  1. Quando Alice riceve una nuova chiave pubblica da Bob, esegue un calcolo DH con la sua chiave privata attuale.
  2. Questo genera un *DH Output* (un segreto condiviso fresco).

3. Successivamente, Alice genera una nuova coppia di chiavi e invia la nuova pubblica a Bob.
4. Bob riceve la chiave, esegue il calcolo e ottiene lo stesso *DH Output*.

Questo processo crea una sequenza di segreti condivisi sempre nuovi, "resettando" la sicurezza del canale.

13.3 La Sintesi: Il Double Ratchet

Il vero potere dell'algoritmo risiede nella combinazione dei due meccanismi: **Double Ratchet = Sym key ratchet + DH ratchet**.

Integrazione dei Componenti

Il diagramma a destra mostra come i due sistemi si alimentano a vicenda:

- **La Root Key (Chiave Radice):** Esiste una chiave suprema chiamata *Root Key*.
- **L'Input del DH Ratchet:** L'output dello scambio Diffie-Hellman (visto nel punto 2) non viene usato direttamente per cifrare i messaggi. Invece, viene inserito in una KDF insieme alla *Root Key* precedente.
- **Generazione delle Chain Keys:** L'output di questa KDF superiore genera:
  - Una nuova *Root Key* (per il futuro passo DH).
  - Una nuova *Chain Key* (Sending/Receiving chain key).

Il Risultato Finale: Post-Compromise Security

Questa architettura risolve il problema evidenziato nella prima slide. Se un attaccante ruba la *Chain Key* attuale:

- Potrà leggere i messaggi per un breve periodo (finché dura quella specifica catena simmetrica).
- Appena l'utente riceve una risposta (che contiene una nuova chiave pubblica DH), il DH Ratchet scatta.
- Viene calcolato un nuovo segreto DH che aggiorna la *Root Key*.
- Viene derivata una catena di chiavi completamente nuova e indipendente dalla precedente.
- L'attaccante è "chiuso fuori" di nuovo (*Self-Healing*).

In sintesi: Il ratchet simmetrico garantisce efficienza e forward secrecy messaggio per messaggio; il ratchet DH garantisce la "guarigione" del protocollo in caso di compromissione, aggiornando la radice di sicurezza tramite nuova matematica asimmetrica.

14 Handshake di TLS 1.3

14.1 1. Introduzione e Obiettivi

L'immagine introduce il handshake della versione 1.3 del protocollo TLS (*Transport Layer Security*). I due obiettivi principali di questa evoluzione rispetto alle versioni precedenti sono sintetizzati chiaramente nel titolo:

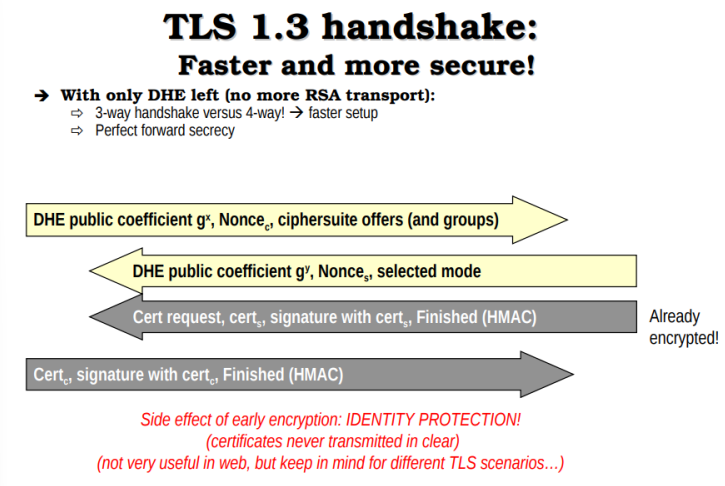

Figure 9: TLS 1.3

- **Velocità ("Faster"):** L'obiettivo è la riduzione della latenza nella connessione e del tempo di setup iniziale.
- **Sicurezza ("More Secure"):** Si punta all'eliminazione di algoritmi obsoleti e all'introduzione di protezioni crittografiche avanzate e moderne.

14.2 2. Cambiamenti Fondamentali nella Crittografia

Il cambiamento architetturale più significativo evidenziato è l'abbandono del trasporto chiavi tramite RSA statico (indicato come *"No more RSA transport"*).

14.2.1 Il passaggio a DHE (Diffie-Hellman Ephemeral)

Il protocollo impone l'uso esclusivo di **DHE** (*Diffie-Hellman Ephemeral*) per lo scambio delle chiavi.

DHE vs RSA: In passato, la chiave di sessione poteva essere cifrata direttamente con la chiave pubblica RSA del server. In TLS 1.3, questo non è più permesso: le chiavi vengono generate matematicamente tramite l'algoritmo Diffie-Hellman utilizzando parametri "effimeri" (temporanei), garantendo che nessuna chiave a lungo termine venga usata per cifrare i dati della sessione corrente.

14.2.2 Vantaggi di questo cambiamento

1. **Handshake a 3 vie (3-way) invece di 4:** Il client ora "indovina" i parametri del gruppo DH e invia la sua parte di chiave immediatamente nel primo messaggio.

Questo riduce il numero di passaggi (*Round Trip Time*) necessari per stabilire la connessione, portando a un setup molto più veloce.

2. **Perfect Forward Secrecy (PFS):** Poiché le chiavi x e y (gli esponenti segreti di Diffie-Hellman) sono effimere e vengono distrutte subito dopo la sessione, si ottiene la *Forward Secrecy*. Anche se un attaccante riuscisse a compromettere la chiave privata a lungo termine del server in futuro, non potrebbe mai decifrare il traffico registrato nel passato, poiché quella chiave non è stata usata per generare la chiave di sessione.

14.3 3. Analisi del Flusso del Handshake

Il diagramma mostra il flusso dettagliato dei messaggi tra Client (sinistra) e Server (destra). È fondamentale notare la transizione dai messaggi in chiaro (frecce gialle) a quelli cifrati (frecce grigie).

14.3.1 Fase 1: Negoziazione (in chiaro)

Questa fase serve a stabilire i parametri crittografici e calcolare la chiave segreta condivisa. I messaggi viaggiano ancora non cifrati.

A. Client → Server (Messaggio "Client Hello") Il client invia:

- **DHE public coefficient g^x:** Il client genera un esponente segreto x e invia immediatamente la sua "quota" pubblica (g^x) per lo scambio Diffie-Hellman. Questo è il tentativo "ottimistico" per velocizzare il processo.
- **Nonce$_c$:** Un numero casuale (usato per prevenire attacchi di replay).
- **Ciphersuite offers:** La lista degli algoritmi di cifratura supportati dal client.
- **(and groups):** I gruppi crittografici supportati per il calcolo DH.

B. Server → Client (Messaggio "Server Hello") Il server risponde con:

- **DHE public coefficient g^y:** Il server genera il suo esponente segreto y, calcola g^y e lo invia al client.
- *Nota Matematica:* A questo punto, avendo scambiato g^x e g^y, entrambe le parti possono calcolare matematicamente la chiave segreta comune (g^{xy}).
- **Nonce$_s$:** Il numero casuale generato dal server.
- **Selected mode:** L'algoritmo di cifratura definitivo scelto tra quelli proposti dal client.

14.3.2 Fase 2: Autenticazione e Finalizzazione (Cifrata)

Nota Cruciale: A partire da questo punto esatto, il traffico nel diagramma è indicato con frecce grigie ed è contrassegnato dalla scritta **"Already encrypted!"**. Questo avviene molto prima rispetto alle versioni precedenti di TLS.

C. Server → Client (Autenticazione) Il server invia i seguenti dati cifrati:

- **Cert Request:** Il server può richiedere il certificato del client (opzionale, usato se è necessaria la mutua autenticazione).
- **Cert$_s$:** Il certificato digitale del server. *Importante:* La chiave pubblica contenuta qui serve ora solo per la verifica della firma, non più per la cifratura dei dati.
- **Signature with cert$_s$:** Il server firma digitalmente l'hash della trascrizione di tutto il handshake avvenuto fino a quel momento. Questo serve ad autenticare il server.
- **Finished (HMAC):** Un codice di autenticazione del messaggio (MAC) che serve a verificare l'integrità dell'intero processo di handshake.

D. Client → Server (Conclusione) Il client conclude inviando (cifrato):

- **Cert$_c$:** Il certificato del client (solo se era stato richiesto dal server).
- **Signature with cert$_c$:** Firma digitale del client per provare il possesso della chiave privata associata al suo certificato.
- **Finished (HMAC):** La conferma finale di integrità da parte del client.

14.4 4. Identity Protection (Protezione dell'Identità)

Si pone una forte enfasi (testo in rosso) su un importante effetto collaterale della cifratura anticipata: la **IDENTITY PROTECTION**.

- **Il Concetto:** Poiché il passaggio dei certificati (`Cert_s` e `Cert_c`) avviene *dopo* che il canale cifrato è stato stabilito (frecce grigie), i certificati non viaggiano mai in chiaro sulla rete.
- **Il Vantaggio:** Un osservatore passivo (Man-in-the-Middle) non può vedere i certificati intercettando i pacchetti. Di conseguenza, non può identificare facilmente l'identità precisa delle parti che stanno comunicando (o quale specifico sottodominio o utente sia coinvolto, al di là dell'indirizzo IP).
- **Nota d'uso:** La slide specifica che questo vantaggio, sebbene tecnicamente rilevante per la privacy, nel contesto del "Web" classico (HTTPS pubblico) potrebbe essere meno utile (dato che il dominio è spesso visibile nel DNS o nel campo SNI in chiaro). Tuttavia, è fondamentale tenere a mente questa caratteristica per altri scenari d'uso di TLS (come VPN, comunicazioni IoT o reti private).

Sintesi Finale

In conclusione, TLS 1.3 ottimizza radicalmente la connessione rendendola:

1. **Più veloce:** Meno passaggi grazie all'invio anticipato e ottimistico di g^x.
2. **Più sicura:** Obbligo di *Forward Secrecy* e cifratura completa dei certificati per proteggere l'identità dei partecipanti.

15 TLS 1.3: L'opzione PSK

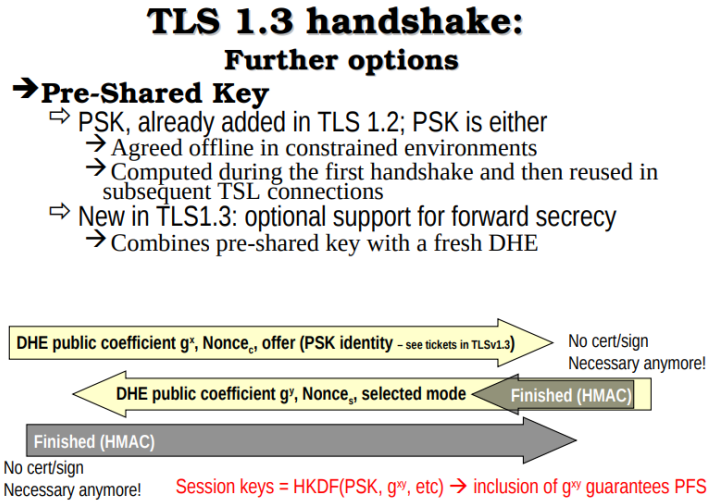

Figure 10: TLS 1.3

L'immagine presenta una modalità alternativa e ottimizzata del handshake TLS 1.3 basata sull'utilizzo di una Pre-Shared Key (PSK), ovvero una chiave pre-condivisa. Sebbene l'uso di PSK fosse già presente in TLS 1.2, la versione 1.3 introduce miglioramenti significativi, in particolare per quanto riguarda la sicurezza futura (Forward Secrecy).

15.1 Origine della Pre-Shared Key (PSK)

La chiave pre-condivisa può essere generata o concordata in due modi distinti:

- **Accordo Offline ("Agreed offline"):**
  - Utilizzato tipicamente in ambienti vincolati (*constrained environments*) come l'IoT (Internet of Things) o dispositivi embedded.
  - In questo scenario, client e server conoscono già la chiave segreta (installata manualmente o tramite un altro canale sicuro) prima di iniziare la connessione.
- **Ripristino di Sessione ("Session Resumption"):**
  - Utilizzato per riconnettersi a un server precedentemente visitato.
  - La PSK viene calcolata durante il primo handshake completo e salvata (sotto forma di Session Ticket).
  - Nei successivi collegamenti, questa chiave viene riutilizzata per evitare un handshake completo, velocizzando notevolmente la connessione.

15.2 L'Innovazione di TLS 1.3: PSK con Forward Secrecy

La novità fondamentale introdotta in TLS 1.3 rispetto alle versioni precedenti è il supporto opzionale per la Perfect Forward Secrecy (PFS) anche durante l'uso di PSK.

Il Meccanismo: Il protocollo combina l'autenticazione basata sulla chiave pre-condivisa (PSK) con un nuovo scambio Diffie-Hellman Ephemeral (DHE).

Obiettivo: Anche se la chiave PSK a lungo termine dovesse essere compromessa in futuro, le sessioni passate rimarrebbero sicure perché protette dalla componente effimera del Diffie-Hellman (g^{xy}).

15.3 Analisi del Flusso del Handshake (PSK + DHE)

Il diagramma illustra come avviene lo scambio dei messaggi. Si noti l'assenza di certificati digitali e firme RSA/ECDSA ("No cert/sign Necessary anymore!").

Fase 1: Client Hello (Offerta) Il Client invia al Server un messaggio contenente:

- **DHE public coefficient g^x:** La parte pubblica effimera del client per lo scambio Diffie-Hellman.
- **Nonce$_c$:** Un numero casuale per evitare replay attack.
- **Offer (PSK identity):** L'identificativo della chiave pre-condivisa che il client desidera utilizzare (spesso riferito ai tickets in TLS 1.3).

Fase 2: Server Hello e Finished Il Server risponde (parte in chiaro e parte cifrata):

- **DHE public coefficient g^y:** La sua parte pubblica effimera.
- **Nonce$_s$:** Il numero casuale del server.
- **Selected mode:** La modalità di cifratura scelta.
- **Finished (HMAC):** (Freccia grigia, cifrato) Il server conferma immediatamente l'integrità del handshake.

Fase 3: Client Finished Il Client conclude (cifrato):

- **Finished (HMAC):** Conferma finale.

Nota sull'Autenticazione: Non sono necessari certificati (Cert) né firme digitali (Sign) perché l'autenticazione è implicita: se entrambe le parti riescono a derivare la stessa chiave di sessione dalla PSK, allora sono autenticate (poiché solo loro conoscono la PSK).

15.4 Derivazione delle Chiavi e Sicurezza Matematica

La parte inferiore dell'immagine descrive la formula critica per la generazione delle chiavi di sessione:

$$\text{Session keys} = \text{HKDF}(\text{PSK}, g^{xy}, \dots)$$

- **HKDF:** È la funzione di derivazione della chiave (HMAC-based Key Derivation Function).
- **Input:** Prende in input sia la chiave pre-condivisa (PSK) sia il segreto condiviso calcolato tramite Diffie-Hellman (g^{xy}).
- **Garanzia di PFS:** La slide sottolinea in rosso che "inclusion of g^{xy} guarantees PFS". L'inclusione del risultato dello scambio Diffie-Hellman (g^{xy}) assicura la Perfect Forward Secrecy. Poiché x e y sono effimeri (distrutti dopo l'uso), un attaccante

non potrà mai ricostruire la chiave di sessione, nemmeno possedendo la PSK in un secondo momento.

Riepilogo Vantaggi

Questo handshake offre il meglio di due mondi:

- **Prestazioni:** È molto veloce (meno passaggi, niente calcoli pesanti di firma digitale o verifica certificati).
- **Sicurezza:** Mantiene la proprietà di Forward Secrecy grazie all'aggiunta dello scambio DHE.

16 New Handshake structure - wrap-up

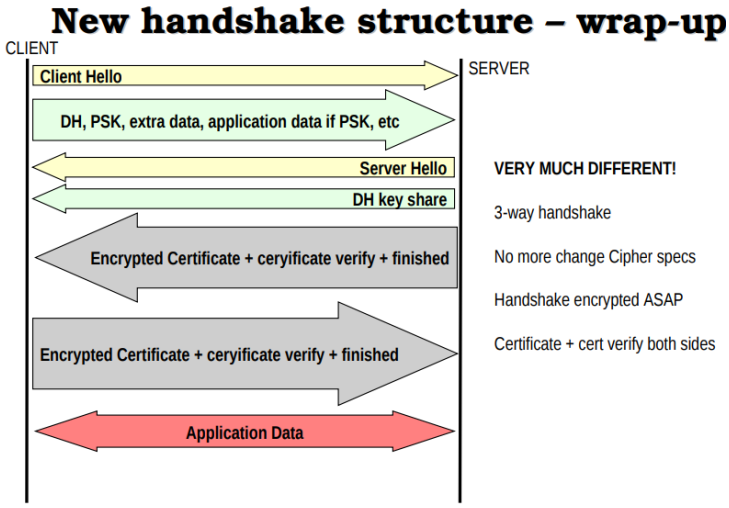

Figure 11: TLS 1.3 New Handshake Structure - Wrap-up

L'immagine illustra il diagramma di flusso completo ("wrap-up") del handshake TLS 1.3. Viene evidenziata la transizione verso un protocollo più efficiente e sicuro, caratterizzato da un ridotto numero di scambi (Round-Trips) e da una cifratura che interviene molto prima nel processo rispetto al passato. Il commento a margine "VERY MUCH DIFFERENT!" sottolinea che non si tratta di un semplice aggiornamento incrementale, ma di una riprogettazione significativa della macchina a stati del protocollo.

16.1 Analisi del Flusso (Step-by-Step)

16.1.1 Fase 1: Negoziazione Iniziale (Client Hello esteso)

A differenza di TLS 1.2, il client non si limita a salutare, ma "indovina" i parametri chiave per velocizzare la connessione.

Messaggio: Client Hello + DH, PSK, extra data...
Dettaglio Tecnico: Il Client invia immediatamente la sua porzione di chiave effimera Diffie-Hellman (Key Share).
Early Data (0-RTT): L'immagine menziona "application data if PSK". Se il client

possiede già una Pre-Shared Key (da una sessione precedente), può inviare dati applicativi cifrati già nel primissimo messaggio, senza attendere la risposta del server (modalità Zero Round Trip Time).

16.1.2 Fase 2: Risposta del Server e Scambio Chiavi

Il Server riceve l'offerta e risponde, completando lo scambio crittografico.

Messaggio: Server Hello + DH key share
Dettaglio Tecnico: Il server seleziona la suite crittografica e fornisce la sua parte pubblica Diffie-Hellman.
Punto di Svolta: Una volta ricevuto questo messaggio, il client e il server possiedono tutto il necessario per calcolare le chiavi di sessione.

16.1.3 Fase 3: Handshake Cifrato (Encrypted Extensions & Auth)

Qui risiede la grande differenza visiva e funzionale (frecce grigie nel diagramma).

Stato: "Handshake encrypted ASAP" (As Soon As Possible).
Messaggio Server: Encrypted Certificate + certificate verify + finished
Tutto ciò che segue lo scambio di chiavi DH è cifrato. Questo include il certificato del server.
Vantaggio Privacy: Un osservatore esterno (eavesdropper) non può più vedere in chiaro quale certificato viene scambiato, proteggendo l'identità del sito (o del client) e rendendo più difficile il fingerprinting.
Messaggio Client: Encrypted Certificate + certificate verify + finished
La slide specifica "Certificate + cert verify both sides". Questo indica una Mutua Autenticazione (mTLS): anche il client sta inviando il proprio certificato per farsi autenticare dal server, ed anche questo avviene in un tunnel cifrato.

16.1.4 Fase 4: Scambio Dati (Application Data)

Concluso il handshake, inizia il flusso bidirezionale dei dati applicativi (freccia rossa), protetto dalle chiavi di sessione derivate.

16.2 Principali Differenze e Innovazioni (Wrap-up)

La sezione destra dell'immagine elenca i cambiamenti architetturali:

16.2.1 3-way Handshake (Ottimizzato):

Il processo è ridotto all'essenziale. In condizioni ideali (1-RTT), il canale sicuro è stabilito in un solo viaggio di andata e ritorno completato.

16.2.2 No more "Change Cipher Specs":

In TLS 1.2, esisteva un messaggio esplicito (ChangeCipherSpec) per segnalare il passaggio alla modalità cifrata, che complicava la macchina a stati e causava vulnerabilità. In TLS 1.3, questo è stato rimosso (o reso un dummy per compatibilità middlebox): la cifratura è implicita subito dopo lo scambio delle chiavi.

16.2.3 Handshake Encrypted ASAP:

Come notato sopra, la protezione crittografica inizia molto prima. La fase di autenticazione (Certificati) non avviene più in chiaro.

16.2.4 Certificate + Cert Verify Both Sides:

Il protocollo gestisce l'autenticazione tramite certificati digitali e messaggi di verifica (Certificate Verify, che contiene una firma digitale su un hash del handshake precedente) sia per il server che, opzionalmente, per il client, garantendo l'integrità totale della negoziazione.

17 TLS 1.3: Handshake e Opzione 0-RTT Data

Una delle innovazioni più significative introdotte con TLS 1.3 è la modalità **0-RTT (Zero Round Trip Time)**, pensata per ottimizzare drasticamente le prestazioni delle connessioni riprese.

17.1 Funzionamento del 0-RTT

La modalità 0-RTT permette al client di inviare dati applicativi *direttamente nel primo messaggio* verso il server, senza attendere il completamento di un handshake completo.

- **Prerequisito:** Questa modalità è utilizzabile solo se client e server hanno concordato una **PSK (Pre-Shared Key)** durante un handshake precedente.
- **Derivazione della Chiave:** Per proteggere questo primo messaggio, la chiave viene derivata utilizzando la funzione HKDF (HMAC-based Key Derivation Function). La formula concettuale è:

  $$\text{Key} = \text{HKDF}_{\text{PSK}}(\text{Client-Hello-content})$$

  In pratica, si "mescola" la PSK con un nuovo nonce, l'offerta (offer) e il coefficiente pubblico DHE (Diffie-Hellman Ephemeral) del client.

17.2 Vantaggi e Casi d'Uso

Il vantaggio principale è la **velocità**.

- Si risparmia un intero RTT (Round Trip Time) rispetto all'handshake standard.
- **Caso d'uso ideale:** È estremamente utile per le richieste HTTP, dove il client può inviare la richiesta (es. GET) nel primissimo pacchetto.
- È indicato per applicazioni che si basano su trasferimenti di dati brevi e frequenti.

17.3 Rischi di Sicurezza

L'uso di 0-RTT comporta un compromesso significativo in termini di sicurezza:

> **Attenzione:** Questa modalità offre una sicurezza inferiore rispetto all'handshake completo ed è intrinsecamente vulnerabile ai **Replay Attack**. Deve essere utilizzata con estrema cautela ("at your own risk").

18 Mitigazione dei Replay Attack in 0-RTT

18.1 Il Problema

La vulnerabilità ai Replay Attack nasce dalla natura stessa del primo messaggio in 0-RTT. La chiave utilizzata per cifrare il primo messaggio **non include il nonce del server**, poiché il client non ha ancora ricevuto alcuna risposta dal server quando invia i dati.

Senza un nonce fornito dal server (che garantisce freschezza), un attaccante può intercettare il messaggio cifrato valido e inviarlo nuovamente al server, il quale potrebbe accettarlo ed elaborarlo una seconda volta (es. inviando nuovamente un ordine di pagamento).

18.2 Tecniche di Mitigazione

Esistono due approcci principali per mitigare questo rischio, anche se non lo eliminano del tutto:

18.2.1 1. Mitigazione basata su Finestra Temporale (Time Window)

Questa tecnica limita il periodo di validità della sessione ripresa.

- Si imposta una vita massima (lifetime) dopo la prima sessione.
- Lo standard suggerisce un default di **7 giorni**.
- **Limitazione:** Non risolve il problema alla radice, ma riduce semplicemente la finestra temporale in cui l'attacco è possibile.

18.2.2 2. Controllo del Riuso dei Nonce (Control Nonce Reuse)

Questa è la soluzione teoricamente più robusta.

- Il server deve mantenere un registro (record) di tutti i nonce inviati dai client.
- Se riceve un nonce già presente nel registro, rifiuta la richiesta come replay.
- **Problemi di Scalabilità:** È molto difficile da implementare in sistemi su larga scala. In architetture con *Load Balancing* o server multipli, mantenere uno stato condiviso e sincronizzato di tutti i nonce in tempo reale è complesso e costoso in termini di performance.

19 Altre caratteristiche e semplificazioni in TLS 1.3

Il principio guida nello sviluppo di TLS 1.3 è stato *"Less is more in security"* (Meno è meglio). Ridurre la complessità riduce la superficie di attacco.

19.1 Rimozione della Rinegoziazione (No Renegotiation)

La rinegoziazione, fonte di molte vulnerabilità nel passato, è stata eliminata.

- Se è necessario cambiare le chiavi all'interno della stessa sessione TLS, si utilizza ora la funzionalità specifica di **Key Upgrade**.

19.2 Crittografia e Gestione Chiavi

- **HKDF:** La funzione *HMAC-based Key Derivation Function* è ora ufficialmente inclusa nello standard per una derivazione delle chiavi più robusta.
- **Curve Ellittiche (EC):** La gestione è stata semplificata. Esiste un singolo formato per i punti EC e non c'è più negoziazione complessa su questo aspetto.

19.3 Rimozione della Compressione

La compressione dei dati a livello TLS è stata rimossa.

- **Motivazione:** Prevenire attacchi di tipo side-channel come **CRIME**, che sfruttavano la compressione per dedurre parti del testo in chiaro (come i cookie di sessione).

19.4 Exported Key (RFC 5705)

TLS 1.3 integra e migliora il meccanismo per esportare materiale crittografico per le applicazioni.

- Permette di generare una chiave segreta condivisa ulteriore (*Exporter master secret*) da utilizzare a livello applicativo.
- **Sicurezza:** Questa chiave è **diversa e indipendente** dalla *session master key*. Se la chiave applicativa viene compromessa, la chiave di sessione TLS rimane sicura.
- **Scopo:** Evita che gli sviluppatori di applicazioni "inventino" i propri metodi (spesso insicuri) per derivare chiavi dal canale sicuro.