26 Certificate

2 dicembre 2025

Indice

1 Il Fallimento della PKI e l'Avvento della Certificate Transparency

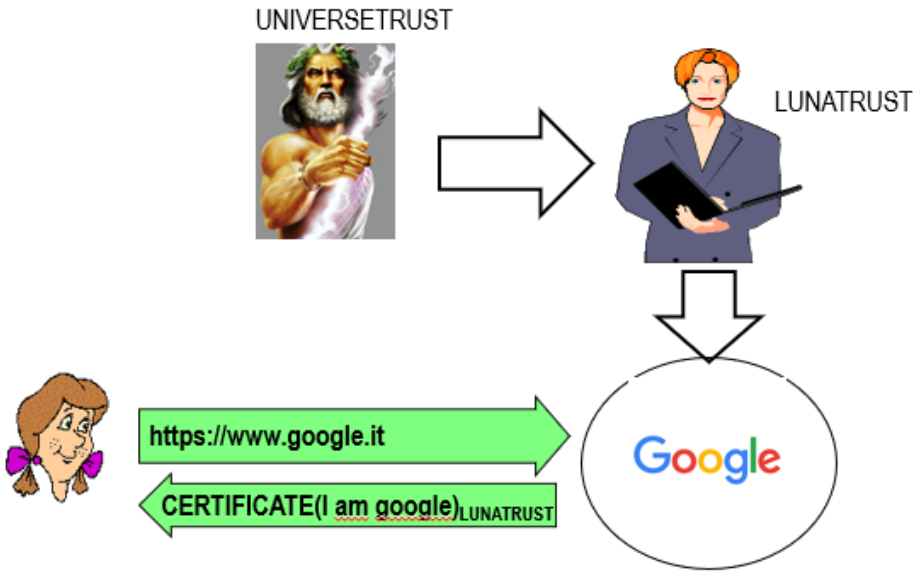
Le slide introducono un punto di svolta critico nella sicurezza del web moderno: la presa di coscienza che il modello tradizionale di Public Key Infrastructure (PKI) basato sulla fiducia cieca nelle Certification Authorities (CA) non è sufficiente.

1.1 Il Pilastro della Sicurezza Web: La Fiducia nelle CA

Fino a pochi anni fa, l'intero modello di sicurezza del web (HTTPS) si basava su un assioma fondamentale: **Le Authority di Certificazione sono fidate**. Come illustrato nello schema della catena di certificati:

- L'utente (o il browser) possiede una lista di *Root CA* pre-installate e fidate (nell'esempio, "UNIVERSETRUST").
- La Root CA delega la fiducia a CA intermedie (nell'esempio, "LUNATRUST").
- La CA intermedia emette il certificato finale per il dominio (es. `www.google.it`).

Il browser accetta il certificato di Google come valido semplicemente perché è firmato matematicamente da "LUNATRUST", di cui si fida tramite la catena che porta a "UNIVERSETRUST".



1.2 Il Problema: I Certificati Falsi (Fake Certificates)

Le slide evidenziano un problema reale molto serio ("A VERY SERIOUS real world problem"): l'emissione di **certificati falsi**. In questo contesto, per "certificato falso" non si intende un certificato con una firma matematica errata, bensì un certificato **crittograficamente valido ma non autorizzato**. Il difetto strutturale della PKI classica è il problema del *Weakest Link* (l'anello debole):

Se una qualsiasi delle centinaia di CA fidate nel mondo viene compromessa (hackerata) o agisce in malafede, essa può emettere un certificato valido per *qualsiasi* dominio (es. Google, Facebook, la tua banca) all'insaputa del legittimo proprietario.

Poiché il browser si fida della CA, accetterà quel certificato falso, permettendo attacchi Man-in-the-Middle su scala globale indistinguibili dalle connessioni legittime.

Concetto Chiave: Il problema dell'"Anello Debole"

Nel sistema PKI tradizionale, i browser si fidano di centinaia di CA in tutto il mondo. Il problema è che tutte hanno lo stesso potere "globale". Se una piccola CA in un paese remoto viene compromessa, può emettere un certificato valido per *google.com*. È come se ogni piccolo ufficio comunale del mondo potesse stampare passaporti validi per il Presidente degli Stati Uniti: basta corromperne uno solo per compromettere l'intera sicurezza globale.

1.3 La Soluzione: Certificate Transparency (CT)

A causa di incidenti reali (come il caso DigiNotar o problemi con Symantec), si è decretato il "fallimento" del vecchio modello PKI basato sulla sola fiducia. La soluzione introdotta è la **Certificate Transparency**.

L'obiettivo della CT è rendere l'emissione dei certificati pubblica e verificabile:

- Ogni certificato emesso deve essere registrato in un *Log pubblico*, accessibile in sola aggiunta (append-only).
- Se un certificato non è presente in questi log pubblici, il browser lo rifiuta.
- Questo permette ai proprietari dei domini (es. Google) di monitorare i log e accorgersi immediatamente se una CA (es. LUNATRUST) ha emesso un certificato a loro nome senza autorizzazione.

Strutture Dati: Merkle Trees Per gestire questi log enormi in modo efficiente e garantire che nessuno possa modificare la storia passata (cancellando tracce di certificati falsi), la Certificate Transparency si basa su strutture dati avanzate basate su hash, specificamente gli **Alberi di Merkle** (Merkle Trees). Questi permettono di verificare l'inclusione di un certificato nel log in modo rapido e sicuro.

2 Evidenze Reali della Crisi: Il Caso dei "Fake Certificates"

La slide "Fact: trusted CA assumption at stake" presenta una serie di incidenti documentati che hanno sgretolato la fiducia cieca nelle Certification Authority. Il problema centrale evidenziato è che, nel modello PKI tradizionale, se una CA (anche piccola o locale) viene compromessa o commette un errore, l'impatto è globale: possono essere generati certificati validi per qualsiasi dominio (es. google.com), permettendo attacchi Man-in-the-Middle invisibili. Vengono distinte due tipologie di incidenti:

2.1 1. Emissione Errata da parte di CA Legittime

In questi casi, CA perfettamente valide e operative hanno emesso certificati fraudolenti per errore umano o procedurale (o, come suggerisce il punto interrogativo nella slide, per negligenza sospetta).

- **TurkTrust (2012):** Una CA turca ha emesso per errore due certificati intermedi a dei clienti finali. Questo ha dato a quei clienti il potere tecnico di generare certificati validi per qualsiasi sito (es. `*.google.com`), bypassando la sicurezza del browser.
- **ANSSI (2013):** L'agenzia nazionale francese per la sicurezza informatica ha emesso un certificato intermedio che è stato poi utilizzato per generare certificati falsi per domini di Google, intercettando il traffico all'interno di una rete privata.

In entrambi i casi, Google ha rilevato (tramite meccanismi di pinning o controlli interni) l'esistenza di certificati validi per i propri domini che però non aveva mai richiesto.

2.2 2. Compromissione di CA Minori (Hacking)

Questo scenario è ancora più grave: attaccanti esterni penetrano nei sistemi di una CA e prendono il controllo delle chiavi di firma.

- **Il Caso DigiNotar (Olanda, 2011):** È l'esempio più eclatante citato ("Dqnotar" nella slide). Hacker iraniani compromisero l'infrastruttura della CA olandese DigiNotar e generarono centinaia di certificati falsi (wildcard) per domini critici come Google, Yahoo, Skype e servizi di intelligence.
- **Conseguenze:** Questi certificati furono usati attivamente per spiare gli utenti (in particolare in Iran). Poiché i browser si fidavano di DigiNotar, l'attacco era completamente trasparente per le vittime. L'incidente portò al fallimento e alla chiusura dell'azienda DigiNotar.
- **DigiCert Sdn. Bhd. (Malesia):** Un altro esempio di CA locale compromessa, distinta dalla nota CA americana DigiCert Inc. (con cui condivide solo parte del nome), che dimostra come la catena di fiducia sia forte quanto il suo anello più debole.

2.3 Conclusione: La Necessità di Trasparenza

Questi eventi dimostrano che la fiducia statica ("Trusted CA Assumption") è un punto debole critico. Non è possibile garantire che centinaia di CA in tutto il mondo siano immuni da errori o attacchi. Questo scenario ha spinto l'industria verso l'adozione della **Certificate Transparency** (introdotta nella slide precedente), un meccanismo per rendere pubblica e verificabile l'emissione di ogni singolo certificato, impedendo che questi incidenti passino inosservati.

3 Un Problema Serio: La Fragilità del Modello PKI

La slide intitolata "A serious problem" analizza le criticità strutturali dell'infrastruttura a chiave pubblica, dimostrando che la sicurezza del modello PKI si sta progressivamente indebolendo ("getting weaker and weaker"). Il problema centrale non è la crittografia in sé, ma la gestione della fiducia. Vengono identificate due macro-categorie di incidenti che minano la sicurezza globale:

3.1 1. Emissione di Certificati "Falsi" ma Validi

Questa categoria riguarda certificati che sono crittograficamente corretti (la firma della CA è valida), ma che sono stati emessi impropriamente per domini di alto profilo (come Google) senza l'autorizzazione dei legittimi proprietari.

- **Il Fenomeno:** Diverse CA di dimensioni medio-piccole hanno rilasciato certificati validi per siti critici ("major sites"). Poiché i browser si fidano di queste CA, i certificati vengono accettati come autentici, permettendo potenziali attacchi Man-in-the-Middle (MITM).
- **Le Cause:** La slide ipotizza tre scenari principali dietro questi eventi:
    1. *Pressione Governativa ("Asked by governments?"):* L'uso di certificati falsi per scopi di sorveglianza di stato o censura.
    2. *Errore Umano ("Mistakenly"):* Errori procedurali o di configurazione da parte degli operatori della CA.

3. *Smarrimento ("Lost"):* Perdita del controllo sulle chiavi di firma.

- **Casi Documentati:**
  - **Turktrust (Dicembre 2012):** Una CA turca ha emesso per errore due certificati intermedi a clienti finali, che avrebbero potuto essere usati per generare certificati falsi per qualsiasi dominio (es. Google).
  - **ANSSI (Dicembre 2013):** L'agenzia governativa francese ha emesso certificati intermedi usati per ispezionare il traffico SSL all'interno di una rete privata, ma che per errore sono "usciti" sulla rete pubblica, minacciando la sicurezza globale.

3.2 2. CA Compromesse (Hacking)

Questa categoria riguarda incidenti in cui l'infrastruttura informatica di una Certification Authority viene violata da attaccanti esterni, che rubano le chiavi private o ottengono il controllo del sistema di emissione.

- **DigiNotar (Olanda, 2011):** Forse il caso più celebre. Hacker sono penetrati nei sistemi della CA olandese DigiNotar emettendo oltre 500 certificati falsi (incluso uno per `*.google.com`). I certificati sono stati usati per intercettare il traffico di utenti iraniani. L'incidente ha portato alla bancarotta della CA.
- **DigiCert Sdn. Bhd. (Malesia):** Un altro esempio di CA locale (da non confondere con la DigiCert americana) i cui sistemi sono stati compromessi, dimostrando che la catena di fiducia è forte quanto il suo anello più debole.

3.3 Conclusione: Un Modello a Rischio

Il testo in rosso in fondo alla slide riassume la gravità della situazione. Il modello PKI attuale deve fronteggiare:

- **Minacce Potenti:** Governi e attori statali con risorse illimitate.
- **Fattore Umano:** Troppi attori (CA) in gioco aumentano la probabilità statistica che qualcuno commetta errori ("make mistakes").

Di conseguenza, il modello di sicurezza basato sulla fiducia incondizionata nelle CA sta diventando insostenibile.

4 Come affrontare le CA malevole: Certificate Transparency

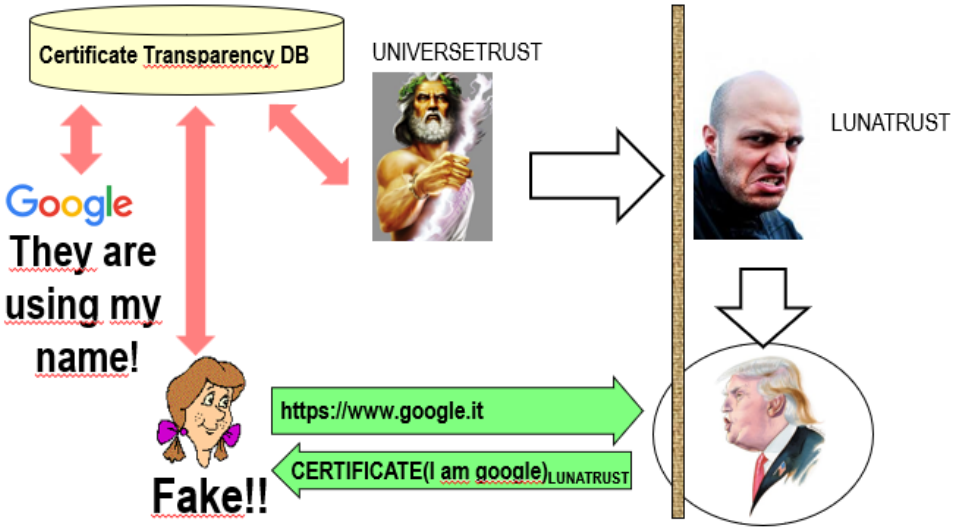
La slide "How to cope with malicious CAs?" presenta la risposta dell'industria (guidata principalmente da Google) al problema delle Certification Authority compromesse o disoneste. La soluzione prende il nome di **Certificate Transparency (CT)**.

4.1 L'Idea Fondamentale

La slide riassume il concetto chiave con una frase semplice ma potente:

> **"Idea: gigantic worldwide DB which anyone can check!"**

Si propone l'istituzione di un sistema di **Log Pubblici** globali. Invece di affidarsi ciecamente alla firma della CA, si richiede che ogni certificato emesso venga registrato in un database pubblico, immutabile e consultabile da chiunque.



4.2 Analisi dello Scenario nella Slide

L'immagine contrappone il comportamento dell'attaccante con i nuovi meccanismi di difesa introdotti dalla CT:

- **L'Attacco (Lato Destro):** Vediamo la CA intermedia "LUNATRUST" (raffigurata come un criminale) che, seppur autorizzata dalla Root CA "UNIVERSETRUST", agisce in malafede. Essa emette un certificato valido per il dominio `https://www.google.it` assegnandolo a un impostore (nella vignetta rappresentato da un noto politico, a indicare un attore terzo non autorizzato).
- **Il Ruolo del Database CT (Lato Sinistro):** Il "Certificate Transparency DB" agisce come un registro mastro pubblico. La sua presenza abilita due controlli fondamentali che prima non esistevano:
  1. **Monitoraggio (Google):** Il legittimo proprietario del dominio (Google) interroga costantemente il DB. Appena LUNATRUST registra il certificato falso, Google se ne accorge immediatamente: *"They are using my name!"*. Questo permette al proprietario di richiedere l'immediata revoca del certificato malevolo prima che faccia troppi danni.
  2. **Verifica (Utente Finale):** L'utente (la ragazza in basso) riceve il certificato da LUNATRUST. Grazie alla Certificate Transparency, il suo browser può verificare se quel certificato è presente e valido nel DB globale. Sapendo che Google non ha autorizzato quella CA o vedendo discrepanze nel log, l'utente può concludere con certezza: **"Fake!!"**.

4.3 Conclusione

La Certificate Transparency risolve il problema della "fiducia cieca" introducendo la **responsabilità (accountability)**. Una CA non può più emettere certificati falsi di nascosto; se lo fa, l'emissione viene registrata pubblicamente, scoperta dal legittimo proprietario e la CA rischia di essere rimossa dai browser (distruggendo il suo business).

5 Gestione Sicura di Grandi Dati: Hash e Fingerprinting

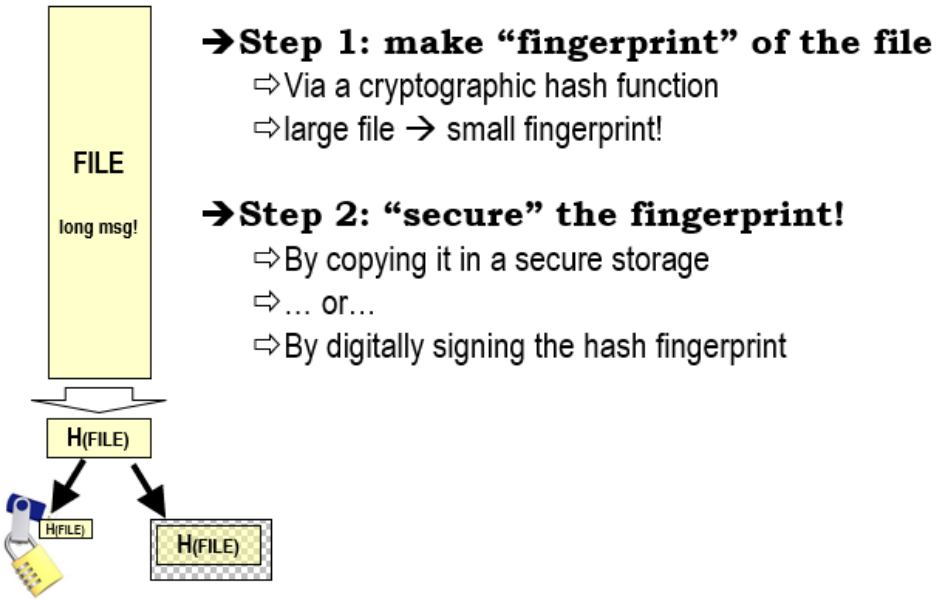
Le slide affrontano il problema tecnico di come implementare e mettere in sicurezza un database gigantesco come quello richiesto dalla Certificate Transparency, introducendo il concetto di strutture dati basate su hash (Merkle Trees) e il principio del "Fingerprinting".

5.1 La Sfida: Un Database Gigantesco

La Certificate Transparency richiede un registro globale, pubblico e verificabile di tutti i certificati emessi al mondo. Come mostrato nella slide "How to implement such gigantic Database?", la domanda è: come possiamo gestire e verificare l'integrità di una mole di dati così massiccia in modo efficiente? La risposta risiede nell'uso di una struttura dati specifica: l'**Albero di Merkle** (Merkle Tree), che verrà approfondito successivamente.

5.2 Il Concetto Base: Mettere in Sicurezza un File

Prima di arrivare agli alberi complessi, la slide "How to secure a file" spiega il principio fondamentale su cui si basa la verifica dell'integrità, ovvero la riduzione del problema dimensionale.

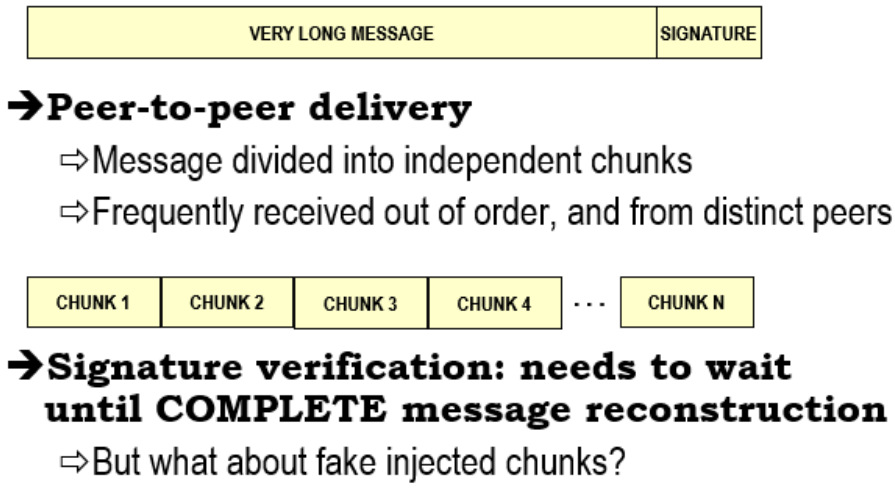


Immaginiamo di voler proteggere un file molto grande ("long msg") senza disporre di uno storage sicuro altrettanto grande. Il processo si divide in due passaggi:

1. **Generazione dell'Impronta (Fingerprint):** Invece di proteggere l'intero file bit per bit, si calcola una sua "impronta digitale" univoca utilizzando una **funzione di Hash Crittografico** ($H(FILE)$).
    - **Proprietà:** Questa operazione trasforma un input di grandi dimensioni in un output di dimensioni fisse e ridotte (es. 256 bit).
    - **Vantaggio:** L'impronta è molto più maneggevole del file originale.
2. **Protezione dell'Impronta:** Una volta ottenuta l'impronta, è sufficiente mettere in sicurezza solo questa piccola stringa di dati. Ci sono due modi principali per farlo:
    - *Secure Storage:* Copiare l'impronta in un luogo sicuro (es. offline, cassaforte). Se l'impronta salvata corrisponde all'hash del file attuale, il file è integro.
    - *Firma Digitale:* Firmare digitalmente l'impronta. Questo garantisce l'autenticità e l'integrità del file originale senza doverlo spostare.

5.3 Il Problema dei Messaggi "Chunked" (Peer-to-Peer)

La slide "What about chunked messages?" estende il problema allo scenario del download Peer-to-Peer (come BitTorrent), dove un file non arriva intero ma diviso in pezzi (*chunks*).



- **Il Problema:** In una rete P2P, i pezzi arrivano disordinati e da fonti diverse (e potenzialmente non fidate).
- **La Limitazione della Firma Unica:** Se avessimo solo la firma dell'intero file (firma alla fine del messaggio), dovremmo aspettare di aver scaricato **tutti** i pezzi e ricomposto il file completo prima di poter verificare se è valido.
- **Il Rischio:** Se anche solo un pezzo fosse corrotto o malevolo ("fake injected chunks"), l'intero download verrebbe invalidato solo alla fine, sprecando banda e tempo. Serve un modo per verificare i pezzi singolarmente man mano che arrivano.

Focus: Richiesta e Verifica di un Certificato

Per capire come questi concetti si applicano alla Certificate Transparency, immaginiamo cosa succede quando il tuo browser richiede un certificato:

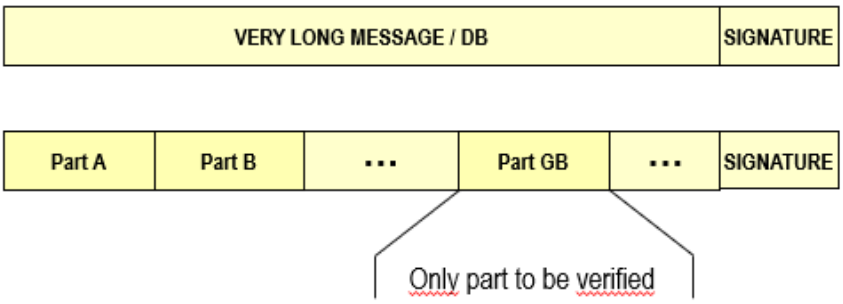
1. **Richiesta:** Il browser si connette a un sito (es. `google.com`). Il server risponde inviando il suo Certificato X.509.
2. **Verifica Tradizionale:** Il browser controlla la firma digitale della CA sul certificato. Questo garantisce che la CA l'abbia emesso, ma non garantisce che la CA sia onesta.
3. **Verifica CT (SCT):** Oltre al certificato, il server invia una prova chiamata **SCT (Signed Certificate Timestamp)**. Questa prova crittografica attesta che "L'impronta (Hash) di questo certificato è stata registrata nel Log Pubblico al momento T".
4. **Sicurezza:** Poiché l'impronta è nel Log pubblico (che usa Merkle Trees per l'immutabilità), Google può vedere che quel certificato esiste. Se fosse falso, Google lo vedrebbe e lo revocherebbe immediatamente. Il browser si fida del certificato non solo perché è firmato, ma perché è *pubblico e monitorato*.

6 Sfide nella Verifica di Integrità: Messaggi Lunghi e Parziali

Le slide esplorano le limitazioni dell'approccio standard basato su firma digitale singola quando si tratta di gestire file di grandi dimensioni, database o trasmissioni a pacchetti (chunked).

6.1 Il Problema della Verifica Parziale

La slide "What about partial verification?" pone un quesito fondamentale per i sistemi distribuiti moderni.



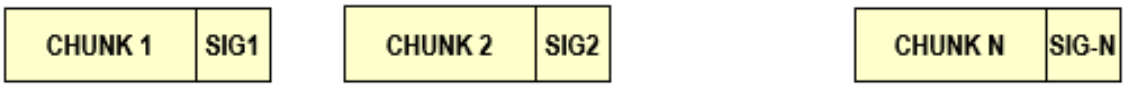
Immaginiamo di avere un database gigantesco ("Very Long Message / DB") firmato digitalmente per intero.

- **Scenario:** Un utente è interessato solo a una piccola porzione dei dati (es. "Part GB").

- **Limitazione:** Con una firma classica calcolata sull'intero file, per verificare l'integrità di quella singola parte, l'utente è costretto a scaricare l'intero database, calcolare l'hash complessivo e confrontarlo con la firma.
- **Inefficienza:** Questo approccio ("need to retrieve the whole database??") è impraticabile per sistemi cloud o distribuiti dove si accede a porzioni di dati su richiesta.

6.2 La Soluzione Ingenua: Firme per ogni Blocco

La slide "Per-chunk signatures?" analizza la soluzione più immediata: dividere il file in blocchi (Chunk 1, Chunk 2, ... Chunk N) e firmare ciascun blocco individualmente.



Sebbene risolva il problema della verifica parziale (posso scaricare e verificare solo il Chunk 2), introduce tre nuovi problemi critici:

1. **Overhead Computazionale:** La firma digitale (crittografia asimmetrica) è computazionalmente costosa.

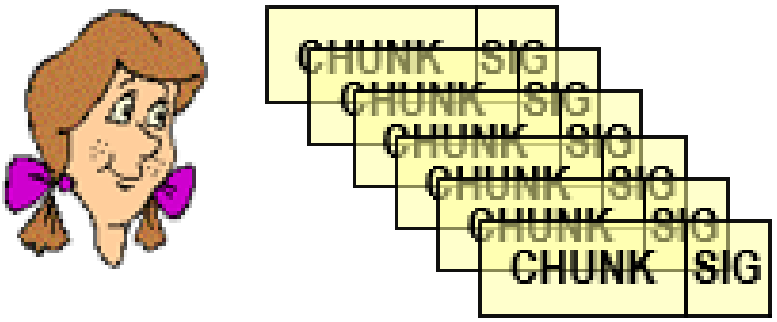
   $$\text{Costo Totale} \approx N_{\text{chunks}} \times (\text{Costo Asimmetrico} + \text{Hash})$$

   Rispetto a una singola firma, il costo esplode linearmente col numero di blocchi. Per milioni di blocchi, è insostenibile.

2. **Overhead di Storage (Spazio):** Le firme digitali occupano spazio significativo.
   - Una firma RSA-2048 occupa 256 byte.
   - Una firma ECDSA (Bitcoin) occupa 64 byte.

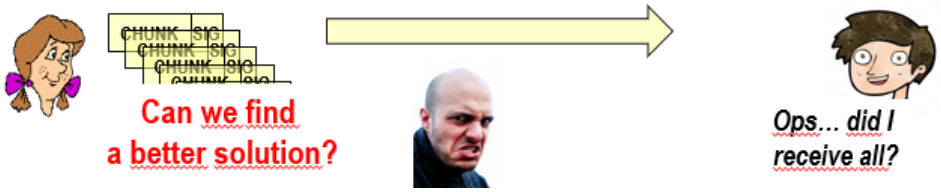
   Se i blocchi sono piccoli (es. 1KB) e sono tantissimi, lo spazio sprecato per memorizzare le firme diventa enorme ("quite a lot if you need MANY of them").

3. **Perdita di Integrità Globale (Strip Attacks):** Questo è il difetto di sicurezza più grave. Se firmo i blocchi separatamente, non esiste un legame che garantisce l'ordine o la completezza del file.



Un attaccante potrebbe:

- Rimuovere un blocco ("Did I receive all?").
- Scambiare l'ordine dei blocchi.



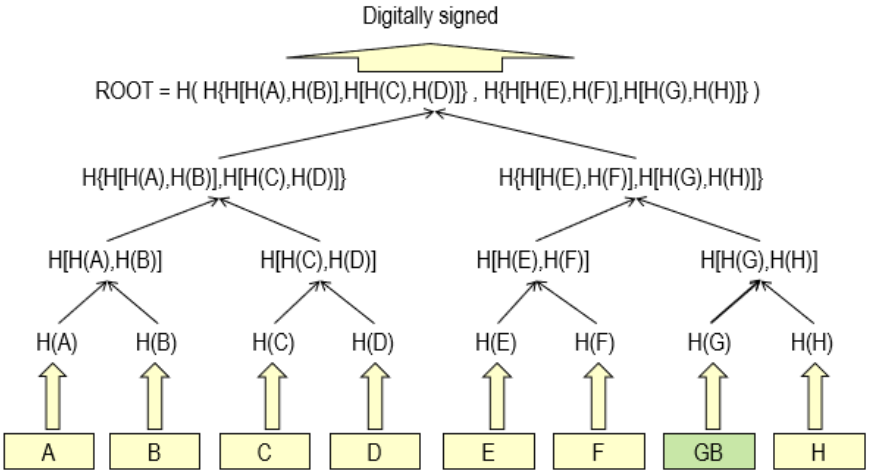
L'utente verificherebbe correttamente le firme dei singoli blocchi ricevuti, ma il file ricostruito sarebbe corrotto o incompleto, senza che se ne accorga. Manca un controllo di integrità per l'intero file ("No more integrity check for whole file").

Conclusione: La soluzione ingenua non funziona. Serve una struttura dati più intelligente che permetta la verifica parziale efficiente senza perdere la visione d'insieme. Questa struttura è l'**Albero di Merkle** (Merkle Tree), che verrà introdotto successivamente.

7 Merkle Trees: Verifica Efficiente dell'Integrità

Le slide introducono l'**Albero di Merkle** (inventato da Ralph Merkle nel 1979) come la soluzione ottimale per verificare l'integrità di singoli elementi all'interno di un grande insieme di dati (come il database della Certificate Transparency), senza dover scaricare l'intero dataset.

7.1 Struttura dell'Albero di Merkle



Un Merkle Tree è un albero binario di hash costruito come segue:

1. **Foglie (Leaves):** I dati reali (nell'esempio: A, B, C, D, E, F, GB, H) si trovano alla base. Ogni foglia è l'hash del dato corrispondente $(H(A), H(B), \dots)$.

2. **Nodi Intermedi:** Ogni nodo intermedio è l'hash della concatenazione dei suoi due figli.

$$\text{Nodo Padre} = H(\text{Figlio Sinistro}||\text{Figlio Destro})$$

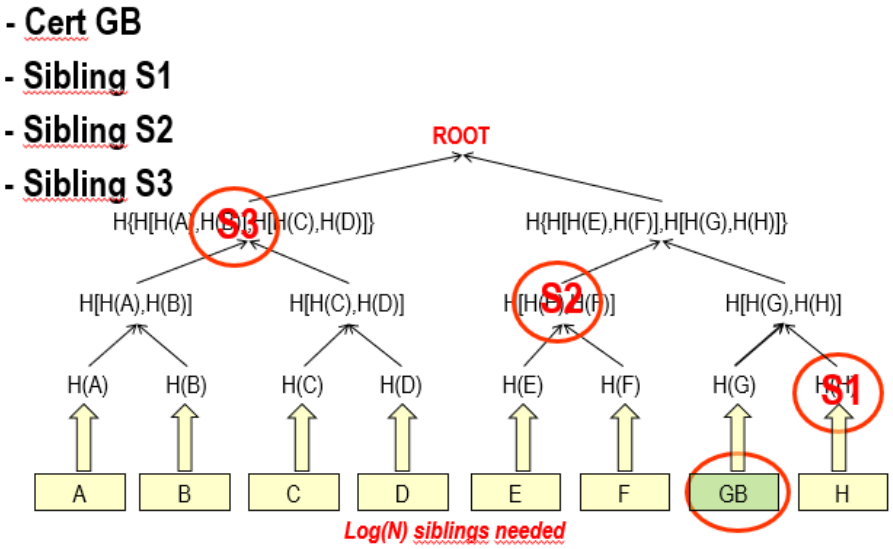
3. **Radice (Root):** Procedendo verso l'alto, si arriva a un unico hash finale chiamato **Merkle Root**.

Proprietà Fondamentale: La Root riassume crittograficamente l'intero albero. Se si modifica anche un solo bit in una qualsiasi foglia (es. A), l'hash $H(A)$ cambia, il che cambia il nodo padre, e così via a cascata fino a cambiare completamente la Root. Pertanto, firmare digitalmente solo la Root ("Digitally signed") equivale a mettere in sicurezza l'intero contenuto dell'albero.

7.2 Verifica di un Singolo Certificato (Proof of Inclusion)

La slide "Single CERT verification" mostra come un utente possa verificare che un dato specifico (es. il certificato "GB") sia presente nell'albero, possedendo solo la Root fidata.

Il concetto di "Siblings" (Fratelli): Per ricalcolare la Root partendo dalla foglia "GB", non servono tutti i nodi dell'albero. Servono solo i nodi "fratelli" lungo il percorso verso la radice.



- Per calcolare il padre di GB, serve il suo vicino: $H(H)$ (indicato come **S1**).
- Per calcolare il nonno, serve il ramo sinistro: $H(H(E)||H(F))$ (indicato come **S2**).
- Per calcolare la Root finale, serve l'intero sotto-albero sinistro (indicato come **S3**).

Il Processo di Verifica: 1. L'utente chiede il certificato "GB". 2. Il server risponde con "GB" e la lista dei *Siblings* ($S1, S2, S3$). 3. L'utente calcola:

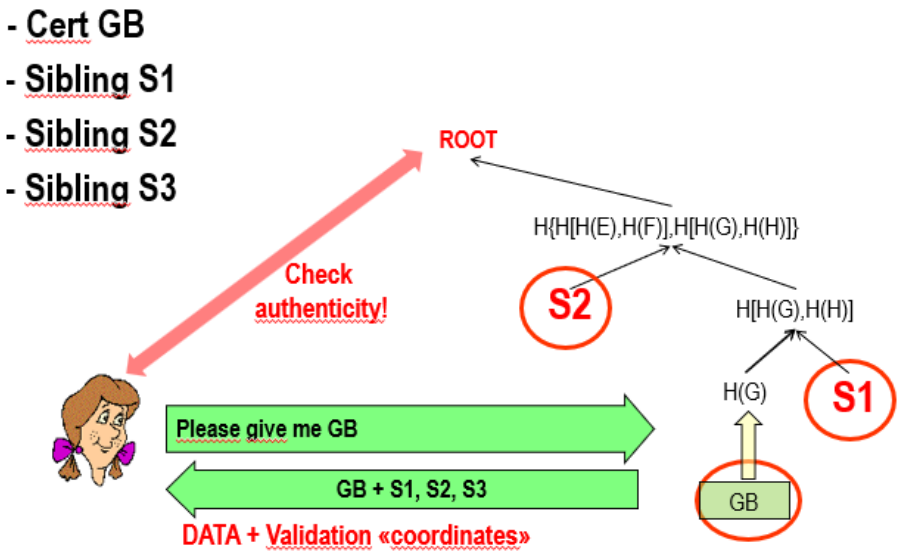
$$\text{Hash}_1 = H(\text{GB})$$

$$\text{Hash}_2 = H(\text{Hash}_1||S1)$$

$$\text{Hash}_3 = H(S2||\text{Hash}_2)$$

$$\text{Root}_{\text{calcolata}} = H(S3||\text{Hash}_3)$$

4. Se $\text{Root}_{\text{calcolata}}$ coincide con la Root firmata nota, allora "GB" è autentico e fa parte dell'albero.



Efficienza: Invece di scaricare N dati, l'utente scarica solo $\log_2(N)$ hash (l'altezza dell'albero). Per un miliardo di certificati, bastano circa 30 hash per la verifica.

Approfondimento: La Potenza dei Logaritmi

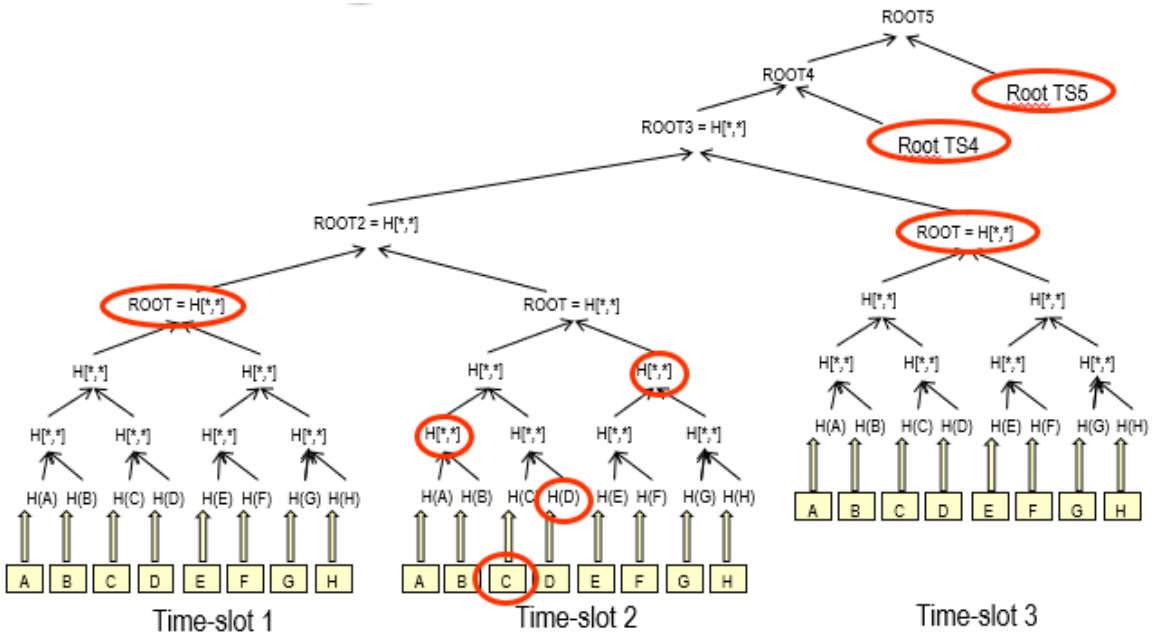
La struttura ad albero di Merkle è il segreto dell'efficienza. Immagina un registro con 1.000.000 di certificati. Per verificare un certificato, invece di scaricare e controllare 1.000.000 di voci, ne devi scaricare solo circa 20 ($\log_2 1.000.000 \approx 20$). Questo rende possibile verificare l'integrità anche su dispositivi limitati come gli smartphone.

8 Merkle Trees nel Tempo e Blockchain

Le slide finali mostrano come gli Alberi di Merkle non siano utili solo per verificare un singolo set statico di dati, ma possano evolvere nel tempo per garantire l'integrità di un registro in continua crescita (come un Log di certificati o una Blockchain).

8.1 Estendere i Merkle Trees nel Tempo (Time-Slots)

La slide "Extending merkle's trees w. time" affronta il problema dell'aggiornamento. Un Log di Certificate Transparency non è statico: nuovi certificati vengono aggiunti continuamente. Non possiamo ricalcolare e rifirmare un intero albero gigante ogni volta che si aggiunge un certificato.



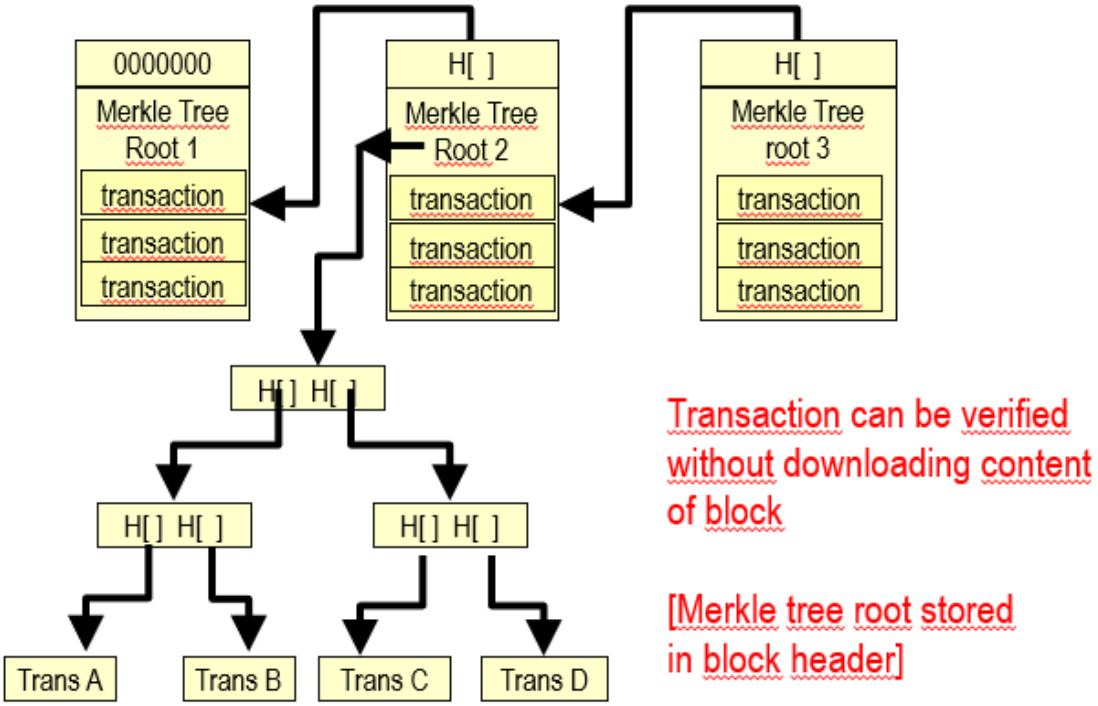
VALIDATION: Siblings for «your» tree + root ot previous timeslot + roots ot next timeslots

La soluzione è strutturare il log in periodi temporali (Time-slots).

- **Struttura:** Ogni Time-slot (es. ogni ora o giorno) ha il proprio Merkle Tree indipendente che congela i certificati emessi in quel periodo.
- **Aggregazione (Tree of Trees):** I Root Hash dei singoli alberi temporali diventano le foglie di un "Super-Albero" (o albero di livello superiore).
- **Vantaggio:** Quando un periodo temporale si chiude, il suo Root Hash è fissato per sempre. Per verificare un dato nel passato, non serve riprocessare i dati futuri.
- **Validazione Complessa:** Per verificare l'inclusione di un certificato C nel Time-slot 2, l'utente ha bisogno di:
  1. I *siblings* interni al proprio albero (Time-slot 2) per arrivare alla Root del periodo.
  2. Le Root degli alberi passati (Time-slot 1) e futuri (Time-slot 3, 4, 5) per ricostruire la Root globale corrente.

8.2 Merkle Trees e Blockchain

L'ultima slide traccia il parallelo con la tecnologia **Blockchain** (es. Bitcoin). Una Blockchain non è altro che una sequenza di blocchi, dove ogni blocco contiene un Merkle Tree delle transazioni.



- **Block Header:** L'intestazione di ogni blocco (la parte che viene "minata" e concatenata) contiene solo la **Merkle Root** delle transazioni di quel blocco, non tutte le transazioni.
- **Verifica Leggera (SPV - Simplified Payment Verification):** Grazie a questa struttura, un client (es. un wallet sul telefono) può verificare che una transazione specifica ("Trans C") è inclusa in un blocco **senza dover scaricare l'intero blocco** (che potrebbe pesare megabyte).
- **Efficienza:** Il client scarica solo gli header della catena (pochi byte) e i *siblings* necessari per ricostruire il percorso dalla transazione alla Merkle Root. Se il calcolo corrisponde alla Root nell'header, la transazione è confermata.

Conclusione: Sia la Certificate Transparency che la Blockchain si basano sulla stessa primitiva crittografica: usare gli hash per condensare grandi moli di dati in un'unica impronta sicura (Root), permettendo verifiche efficienti e a prova di manomissione.

9 Applicazioni degli Alberi di Merkle

La slide conclusiva "Merkle Trees: Applications" offre una panoramica sull'ubiquità di questa struttura dati nella crittografia moderna e nei sistemi distribuiti. Grazie alla loro capacità di garantire l'integrità dei dati in modo efficiente e scalabile, i Merkle Trees sono diventati un componente fondamentale in diversi ambiti. Le principali applicazioni citate sono:

9.1 1. Blockchain e Criptovalute

L'applicazione più famosa è senza dubbio nel mondo delle **Blockchain** (come Bitcoin ed Ethereum).

- **Efficient Storage:** Come visto nella sezione precedente, ogni blocco della catena contiene la Merkle Root di tutte le transazioni in esso incluse. Questo permette di "riassumere" migliaia di transazioni in una stringa di 32 byte (l'hash della radice).
- **Verifica Leggera (Light Clients):** Permette ai nodi che non possiedono l'intera blockchain (es. portafogli su smartphone) di verificare se una transazione è stata inclusa in un blocco scaricando solo una piccola prova (i *siblings*), invece di gigabyte di dati.

9.2 2. Certificate Transparency (Google)

Come discusso ampiamente nelle sezioni precedenti, i Merkle Trees sono il cuore tecnologico dei **Log Pubblici** per i certificati web.

- La struttura ad albero permette di inserire milioni di certificati in un log append-only.
- I monitor (come Google) possono verificare rapidamente che nessun certificato sia stato modificato o rimosso retroattivamente, garantendo l'integrità globale della PKI.

9.3 3. Validazione di File Frammentati (File Chunk Validation)

Nei sistemi di file sharing distribuiti (come BitTorrent, IPFS o sistemi di aggiornamento software P2P):

- I file di grandi dimensioni vengono divisi in pezzi (*chunks*).
- Invece di firmare ogni pezzo, si costruisce un Merkle Tree dei pezzi e si firma solo la radice.
- Quando un utente scarica un pezzo da un peer sconosciuto, può verificarne l'integrità immediatamente confrontando il suo hash con il percorso verso la radice firmata, proteggendosi da dati corrotti o malevoli.

9.4 4. Altre Applicazioni Avanzate

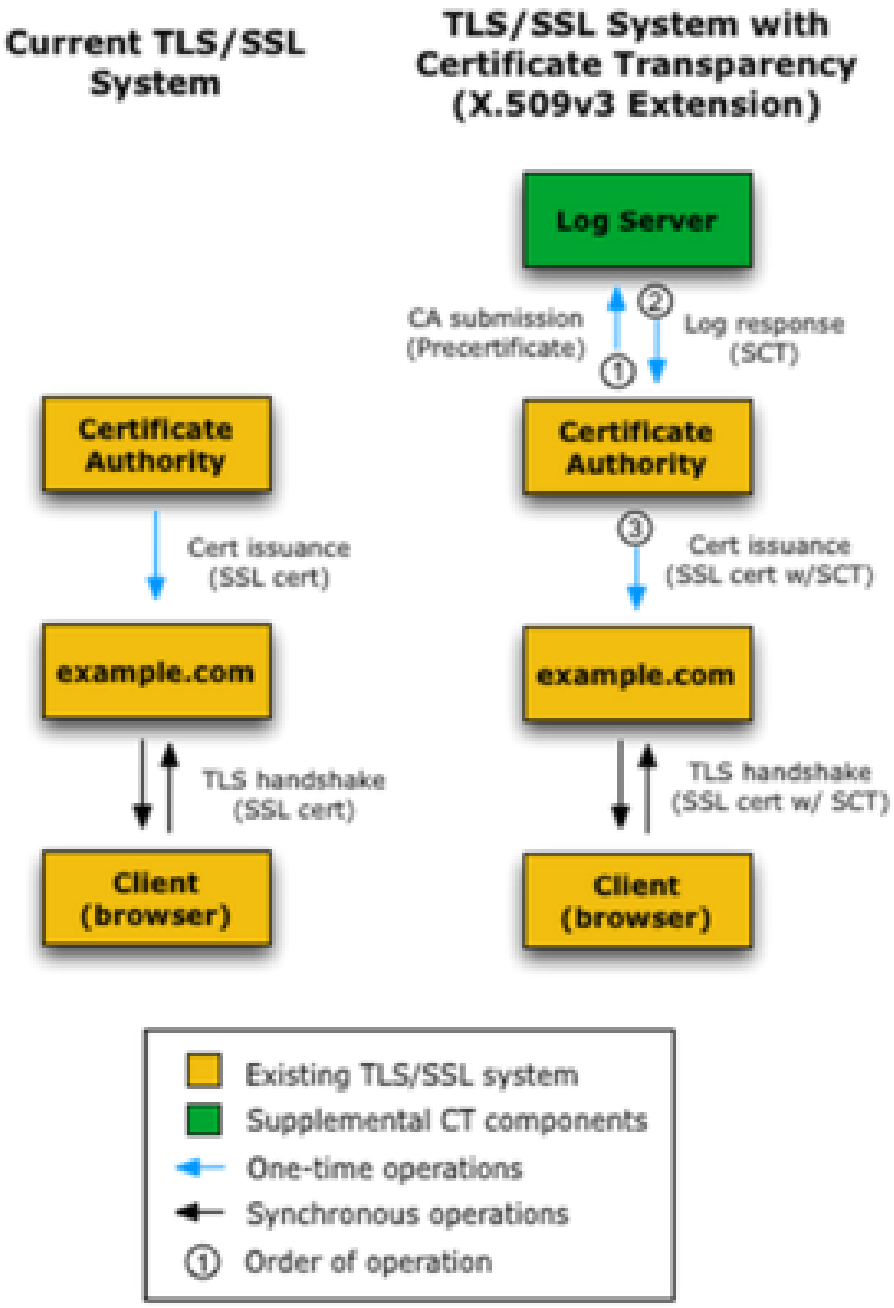
La slide elenca ulteriori utilizzi tecnici che dimostrano la versatilità della struttura:

- **One-time signatures:** Schemi di firma come *Lamport* o *Winternitz* usano Merkle Trees per aggregare molte chiavi "usa e getta" in un'unica chiave pubblica gestibile.
- **Autenticazione di Memoria e Storage:** Usati in hardware sicuro per verificare che la memoria non sia stata manomessa fisicamente.
- **Node Authentication:** In reti distribuite per autenticare i partecipanti in modo efficiente.

In sintesi, ovunque ci sia la necessità di verificare l'integrità di grandi insiemi di dati in modo efficiente, distribuito e sicuro, è molto probabile trovare un albero di Merkle.

10 Certificate Transparency in a Nutshell: Come Funziona?

La slide "Certificate Transparency in a nutshell" illustra il flusso tecnico modificato per l'emissione e l'utilizzo dei certificati, confrontando il sistema tradizionale con quello abilitato alla CT.



10.1 1. Fase di Emissione (Issuance)

Nel nuovo modello, quando una CA deve emettere un certificato per un dominio (es. example.com), deve compiere un passaggio aggiuntivo obbligatorio.

1. **Invio al Log:** La CA invia il "pre-certificato" a un Log Server pubblico ("Add it to the log server").

2. **Ricezione SCT:** Il Log Server registra l'evento e restituisce una ricevuta crittografica chiamata **SCT (Signed Certificate Timestamp)**. Questa ricevuta è la prova che il certificato sarà reso pubblico entro un tempo prestabilito (es. 24 ore).

3. **Inclusione nel Certificato:** La CA incorpora l'SCT all'interno del certificato finale (come estensione X.509) e lo consegna al proprietario del sito.

10.2 2. Fase di Utilizzo (Handshake TLS)

Quando un utente (browser) si collega al sito:

- Il server invia il certificato contenente l'SCT.
- Il browser verifica la validità standard del certificato (firma, date, ecc.).
- **Verifica Aggiuntiva:** Il browser verifica anche la presenza e la validità dell'SCT. Se l'SCT manca o non proviene da un Log fidato, il browser può rifiutare la connessione o mostrare un avviso, poiché il certificato non è "trasparente".

10.3 Perché la Certificate Transparency?

La slide successiva ("Why certificate transparency?") riassume gli obiettivi strategici di questo sistema:

- **Visibilità Totale:** L'obiettivo primario è rendere *impossibile* per una CA emettere un certificato valido per un dominio (es. Google) senza che il proprietario del dominio ne venga a conoscenza. Se la CA emette il certificato ma non lo logga, il browser non lo accetta (manca SCT). Se lo logga, il proprietario lo vede nei log pubblici.
- **Consistenza Globale:** Un secondo obiettivo cruciale è garantire che tutti vedano la stessa lista di certificati ("Make sure that we all see the SAME list").
  - Questo previene l'attacco **"Split World"** (Mondo Diviso), dove un Log Server corrotto potrebbe mostrare una versione "pulita" del log al pubblico e una versione con certificati falsi solo alla vittima.
  - Le strutture dati usate (Merkle Trees e Signed Tree Heads) permettono di dimostrare matematicamente che la visione del log è consistente per tutti gli utenti nel mondo.

11 Certificate Transparency: Storia e Dettagli Tecnici

Le slide ripercorrono la storia della Certificate Transparency (CT) e spiegano il funzionamento dei log basati sugli alberi di Merkle che ne costituiscono il cuore.

11.1 Origini e Standardizzazione

La Certificate Transparency non è nata come standard ufficiale, ma come iniziativa privata di Google nel 2013, in risposta agli incidenti di sicurezza delle CA.

- **2013:** Google lancia il progetto e pubblica la prima specifica sperimentale (RFC 6962).
- **2014:** L'IETF (Internet Engineering Task Force) adotta il progetto creando il gruppo di lavoro "trans" per standardizzarlo ufficialmente (RFC 6962-bis).
- **Adozione:** Oggi è integrata in tutti i principali browser (Chrome, Firefox, Safari) ed è obbligatoria per molti tipi di certificati.

11.2 Come Funziona Tecnicamente: Append-Only Log

Il meccanismo principale è un Log Pubblico che utilizza i **Merkle Trees** in una configurazione specifica chiamata "Chron Tree" o **Append-Only Tree**.

11.2.1 Proprietà Append-Only

La caratteristica fondamentale per la sicurezza è che il log può solo crescere aggiungendo nuovi elementi alla fine. Non è possibile:

- Modificare un certificato già inserito.
- Eliminare un certificato dalla storia.
- Inserire un certificato "nel passato" (retroattivamente).

Ogni voce è marcata temporalmente (*Timestamped*).

11.2.2 Evoluzione dell'Albero (Figure 1 e 2)

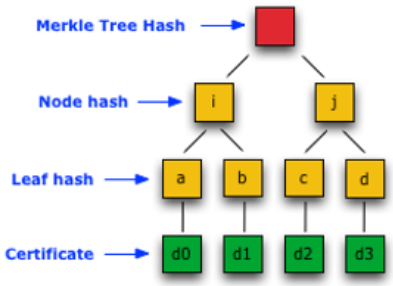


Figure 1

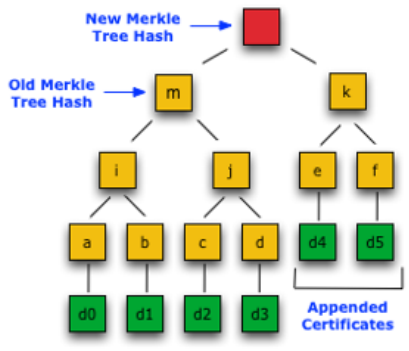


Figure 2

Le immagini mostrano come l'albero cresce mantenendo la sua integrità crittografica:

- **Figura 1 (Stato Iniziale):** Abbiamo 4 certificati ($d0 \dots d3$). L'albero ha una certa Root Hash (quadrato rosso).

- **Figura 2 (Aggiornamento):** Vengono aggiunti due nuovi certificati ($d4, d5$). Invece di ricostruire tutto da zero, il nuovo albero incorpora il vecchio come sotto-albero sinistro. La "Old Merkle Tree Hash" diventa un nodo figlio (m) del nuovo albero più grande.
- **Verifica di Consistenza:** Grazie a questa struttura, è possibile generare una *Consistency Proof* che dimostra matematicamente che il "Nuovo Albero" contiene interamente il "Vecchio Albero" senza aver modificato nulla dei dati preesistenti.

12 Merkle Consistency Proof: Garantire l'Immutabilità della Storia

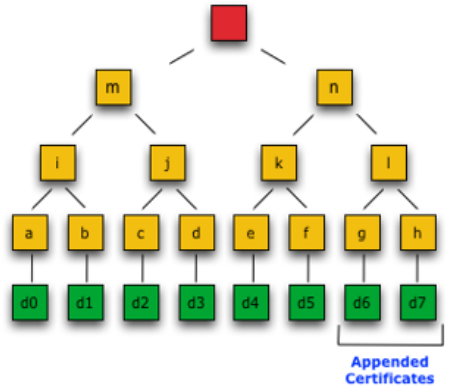


Figure 3

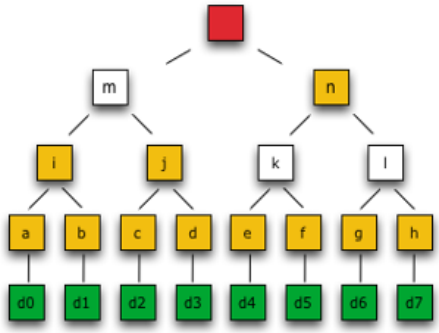


Figure 4

La slide "Merkle Consistency proof" illustra il meccanismo crittografico utilizzato per verificare che un Log (registro) si sia evoluto onestamente nel tempo. Dato che un Log di Certificate Transparency è una struttura *append-only* (solo accodamento), è necessario un metodo per provare matematicamente che una versione successiva dell'albero (Log al tempo T_2) sia una perfetta estensione di una versione precedente (Log al tempo T_1).

12.1 Obiettivo della Prova di Consistenza

L'obiettivo è verificare la consistenza tra due istantanee (snapshot) del Log. Se al tempo T_1 il Log aveva N foglie e una Root Hash R_1, e al tempo T_2 ha $N + K$ foglie e una Root Hash R_2, la prova deve garantire che:

1. **Inclusione Totale:** Il nuovo albero include tutte le foglie del vecchio albero.
2. **Ordine e Integrità:** Le foglie vecchie sono nello stesso identico ordine e non sono state modificate.
3. **Nessun Back-dating:** I nuovi certificati sono stati aggiunti solo alla fine (append), non inseriti "in mezzo" alla storia passata.

12.2 Funzionamento Tecnico

Non è necessario scaricare l'intero database per fare questa verifica. Sfruttando la struttura dell'Albero di Merkle, la *Consistency Proof* è costituita da una piccola lista di nodi hash (le radici dei sottoalberi). Guardando le figure nella slide:

- **Figura 3 (Stato Corrente):** Mostra l'albero completo con 8 foglie ($d0 \dots d7$). I certificati $d6$ e $d7$ sono quelli appena aggiunti ("Appended Certificates").
- **Figura 4 (Logica della Prova):** Per provare che questo albero è consistente con una versione precedente (ad esempio, una che conteneva solo le foglie da $d0$ a $d5$), il server fornisce una prova costituita da nodi specifici (come m, k, l).

Il concetto chiave: La prova di consistenza permette al client di prendere la **Vecchia Root Hash** (che aveva memorizzato in precedenza) e ricalcolarla utilizzando un sottoinsieme di nodi del **Nuovo Albero**. Se il calcolo ha successo e i nodi usati sono validi nel nuovo albero, allora è matematicamente certo che la "vecchia storia" è contenuta intatta nella "nuova storia".

> In sintesi: La Consistency Proof dimostra che il Log non ha mai riscritto il passato (modificando o cancellando certificati) e che l'unica operazione eseguita è stata l'aggiunta di nuovi dati.

13 Merkle Audit Proof: Verifica e Problemi di Privacy

La slide "Merkle Audit Proof" analizza come un client (es. un browser) possa verificare matematicamente che un certificato specifico sia incluso nel Log pubblico, e quali implicazioni di privacy nascano da questa operazione.

13.1 Il Meccanismo di Audit (Inclusion Proof)

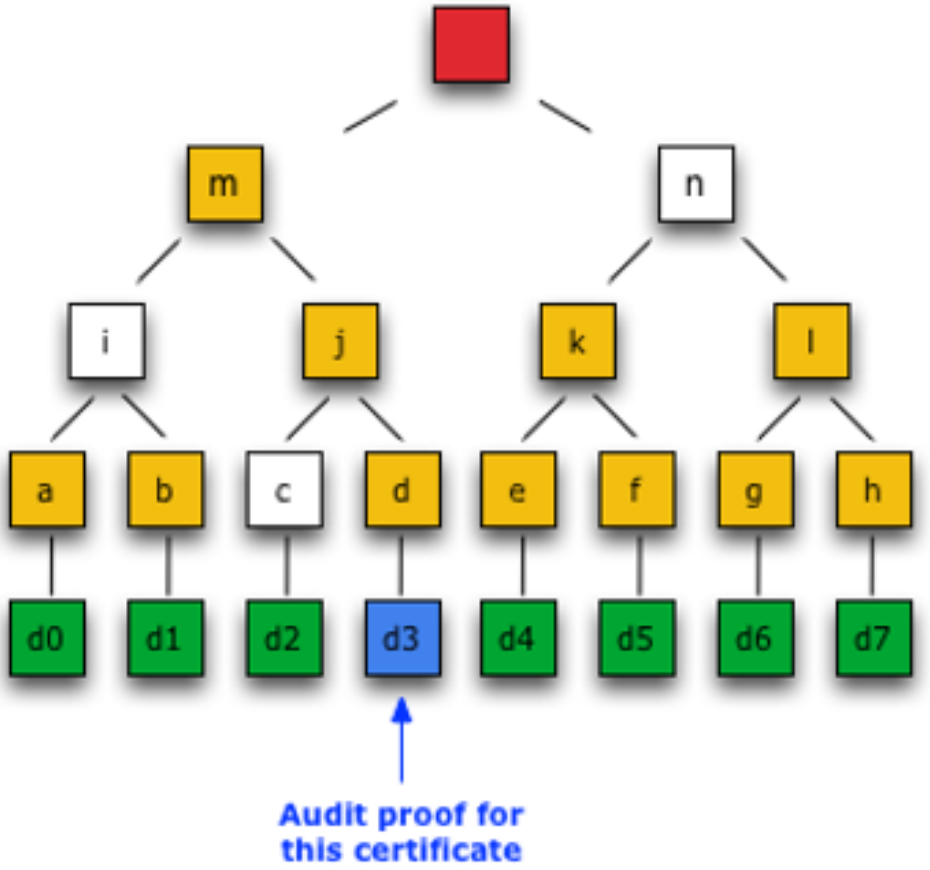


Figure 5

Guardando la **Figura 5**, l'obiettivo è dimostrare che il certificato rappresentato dalla foglia **d3** (blu) è parte dell'albero che ha come radice il nodo rosso. Come visto per i Merkle Trees, non è necessario scaricare l'intero albero. Sono necessari solo i nodi "fratelli" (siblings) lungo il percorso verso la radice. Per calcolare la Root partendo da $d3$, servono i nodi evidenziati in giallo/bianco necessari per le operazioni di hash:

1. Ho $d3$. Mi serve il fratello **c** per calcolare il padre $j = H(c||d3)$.
2. Ho j. Mi serve il fratello **i** per calcolare il nonno $m = H(i||j)$.
3. Ho m. Mi serve il fratello **n** (che copre tutto il sottoalbero destro) per calcolare la **Root** $= H(m||n)$.

Se la Root calcolata coincide con quella firmata dal Log Server, la prova è superata.

13.2 Il Problema della Privacy: "Visible to All"

La slide solleva una criticità fondamentale ("Issue"). Per ottenere i nodi fratelli (c, i, n), il client deve chiederli al Log Server.

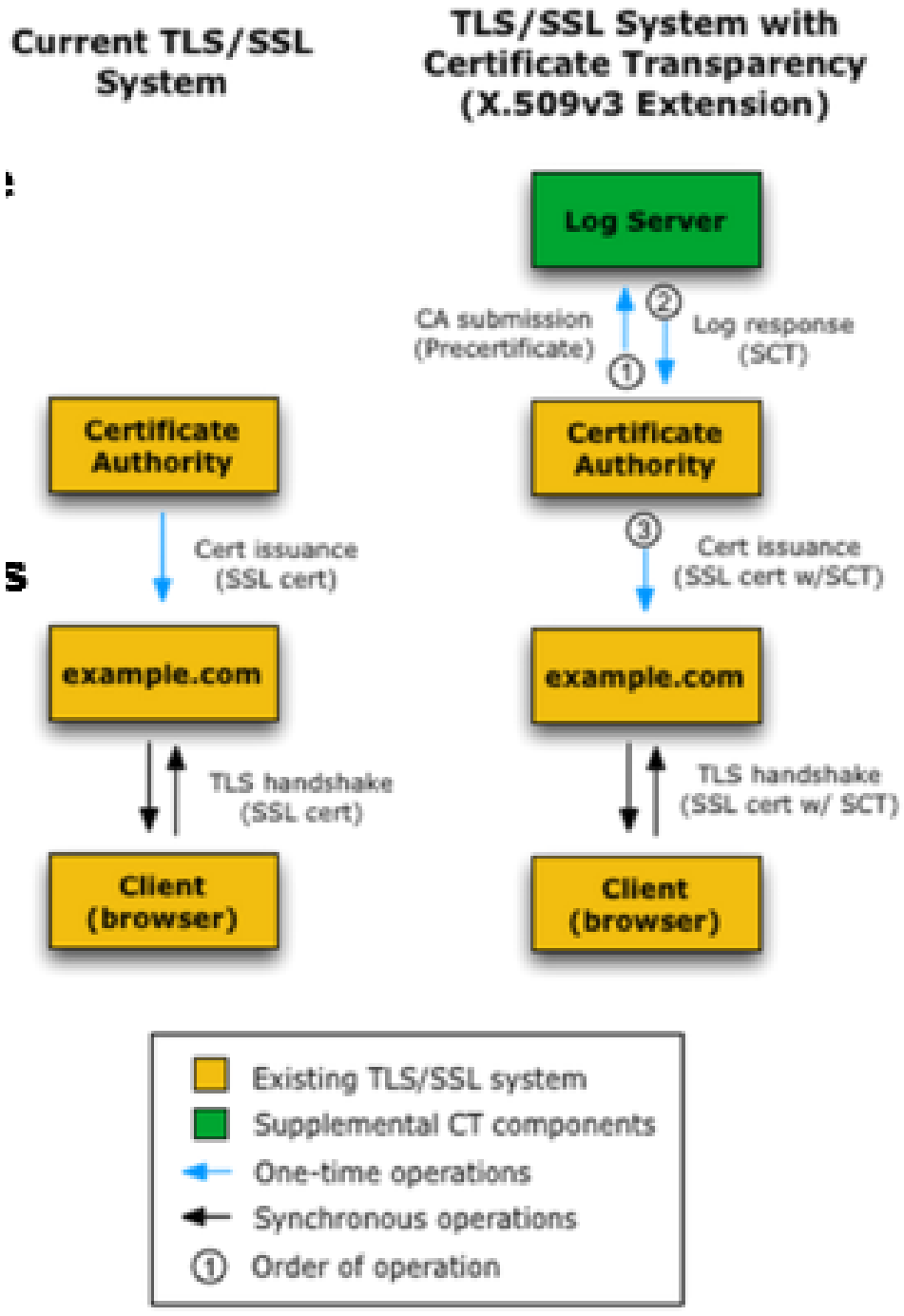
- **La Richiesta:** "Ciao Log Server, mi dai i fratelli per verificare il certificato di *www.dissidenti-politici.com*?"
- **La Minaccia:** Facendo questa richiesta, il client rivela al Log Server esattamente quale sito sta visitando. Poiché l'operazione è online, si crea una traccia dei comportamenti di navigazione dell'utente.
- **Big Brother:** La frase "Remember who are the threats today" allude al fatto che i Log Server potrebbero essere gestiti da grandi corporazioni (Google) o monitorati da agenzie governative, trasformando un meccanismo di sicurezza in uno strumento di sorveglianza di massa.

13.3 L'Alternativa: Protocolli di Gossiping

Per mitigare questo rischio, la slide suggerisce l'uso di protocolli di **Gossiping** (pettegolezzo). Invece di chiedere direttamente al Log Server "verificami questo sito specifico", i client si scambiano informazioni tra loro o scaricano porzioni di prove in modo casuale o aggregato, in modo da offuscare la richiesta specifica e rendere indistinguibile il traffico di verifica dal rumore di fondo. Questo approccio (ancora in fase di sviluppo/standardizzazione nel 2015) mira a disaccoppiare la sicurezza (verifica dell'integrità) dalla privacy (non tracciabilità).

14 Certificate Transparency nel Mondo Reale

La slide "Certificate Transparency in the real world" illustra lo stato dell'arte dell'adozione di questa tecnologia e descrive il flusso operativo standardizzato per l'emissione di certificati tracciabili.



14.1 Storia e Standardizzazione

La Certificate Transparency (CT) non è rimasta un concetto teorico, ma è diventata uno standard di fatto per la sicurezza web.

- **Origine:** L'iniziativa è stata lanciata da Google nel **luglio 2013**.
- **Standard IETF:** La specifica tecnica è stata formalizzata nella **RFC 6962** (inizialmente sperimentale) sotto la supervisione del gruppo di lavoro IETF "trans". Questo ha garantito che il protocollo non fosse proprietario ma aperto e interoperabile.

- **Adozione:** Oggi CT è integrata in tutti i principali browser ed è supportata da grandi provider (PayPal, CertSign). Esistono molteplici "Log Servers" operativi gestiti da entità diverse come Google, Cloudflare e DigiCert, garantendo decentralizzazione e robustezza.

14.2 Il Flusso Operativo: Confronto

La parte destra della slide mostra un confronto visivo tra il vecchio sistema e quello nuovo.

14.2.1 1. Sistema TLS/SSL Classico (Legacy)

Nel modello tradizionale (colonna sinistra del diagramma):

1. La **CA** emette il certificato per il dominio (es. *example.com*).
2. Il server web invia questo certificato al **Client** (browser) durante l'handshake.
3. *Problema:* Nessuno, a parte la CA e il Server, sa che questo certificato esiste. Se è fraudolento, passa inosservato.

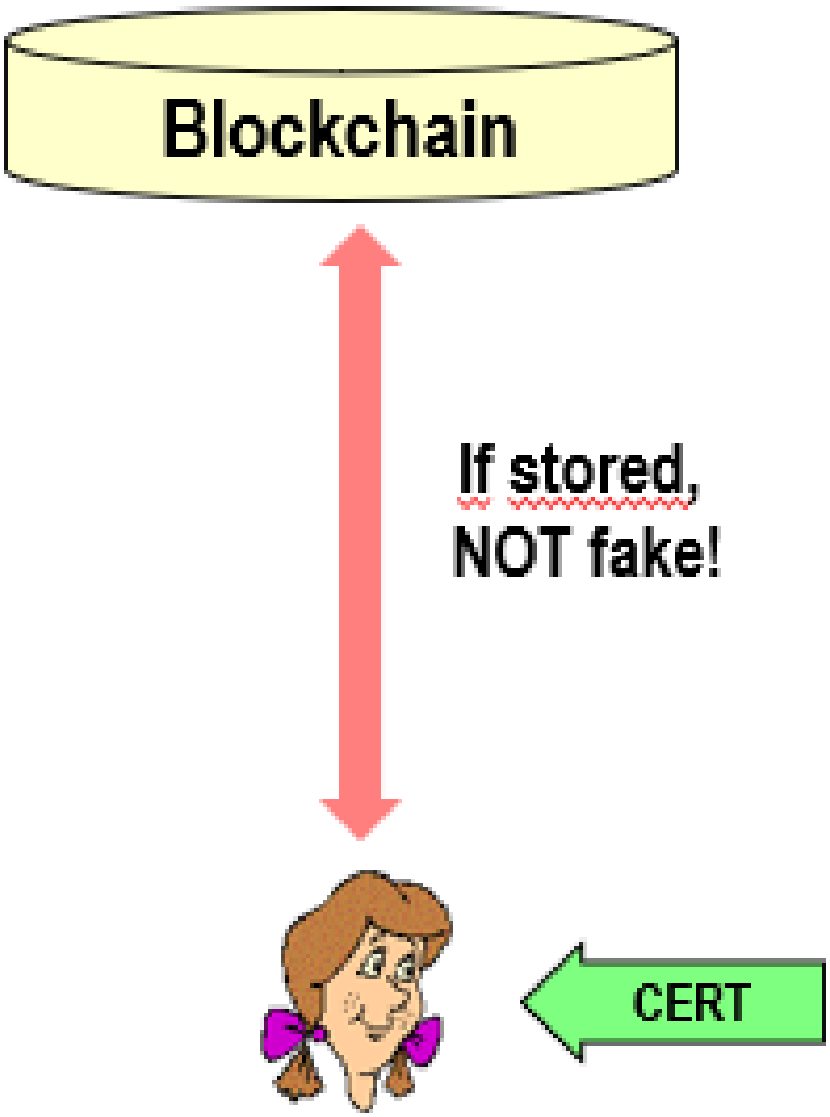
14.2.2 2. Sistema con Certificate Transparency

Nel modello CT (colonna destra del diagramma), il processo di emissione cambia per includere la prova di pubblicazione:

1. **Pre-Certificato:** La CA invia una bozza del certificato ("Precertificate") a un **Log Server** pubblico.
2. **Ricevuta SCT:** Il Log Server registra la richiesta e risponde immediatamente con un **SCT (Signed Certificate Timestamp)**. Questa è la "ricevuta" crittografica che prova che il certificato verrà aggiunto al registro pubblico entro un tempo garantito (es. 24 ore).
3. **Emissione Finale:** La CA incorpora l'SCT all'interno del certificato finale (utilizzando un'estensione X.509v3) e lo consegna al proprietario del sito (*example.com*).
4. **Handshake TLS:** Quando il Client si collega, il server invia il certificato che contiene l'SCT. Il browser verifica la presenza dell'SCT: se c'è, significa che il certificato è pubblico e monitorabile; se manca, il certificato potrebbe essere rifiutato (specialmente in Chrome/Safari).

15 Certificate Transparency vs Blockchain: Un Malinteso Comune

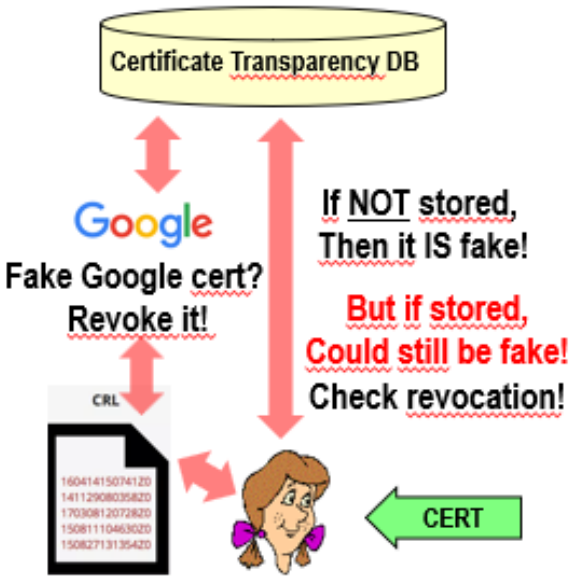
Le slide affrontano una domanda frequente e insidiosa: *"La Certificate Transparency è una Blockchain?"*. Sebbene le due tecnologie condividano le stesse fondamenta crittografiche (gli Alberi di Merkle), la risposta è un netto **NO**. Le slide analizzano le somiglianze architetturali e le differenze filosofiche fondamentali.



15.1 Somiglianze Architetturali

È innegabile che le due tecnologie si assomiglino ("It definitely looks like"):

- **Struttura Dati:** Entrambe utilizzano un registro *append-only* basato su Merkle Trees. L'architettura del ledger è quasi identica a quella di Bitcoin.
- **Distribuibilità:** Teoricamente, la CT potrebbe essere implementata come un registro distribuito con meccanismi di consenso, proprio come una Blockchain, invece di avere Log Server gestiti da singole entità (come Google o Cloudflare).



15.2 La Differenza Cruciale: Validità vs Visibilità

Il punto di divergenza fondamentale risiede nella semantica di ciò che viene memorizzato ("But it is NOT a Blockchain"):

1. **Blockchain (Consenso sulla Validità):** In una blockchain (es. Bitcoin), l'inclusione di una transazione nel blocco implica la sua **validità**. I nodi rifiutano le transazioni invalide (es. spendere soldi che non si hanno). Se è nella catena, è vero.

2. **Certificate Transparency (Log di Visibilità):** Un Log CT accetta e registra **qualsiasi** certificato emesso da una CA riconosciuta, anche se quel certificato è "falso" (es. emesso da una CA compromessa per un dominio che non l'ha richiesto). Il Log non verifica se il certificato è "autorizzato", ma solo che è stato emesso.

15.3 Il Modello di Sicurezza

L'immagine nella slide (con le frecce rosse) illustra come la sicurezza venga garantita in CT, differenziandola dal modello "preventivo" della Blockchain.

- **Se NON è nel Log:** Il certificato è scartato dal browser ("Fake!"). La presenza nel log è un requisito tecnico per l'uso.

- **Se È nel Log:** Il certificato potrebbe comunque essere malevolo ("Could still be fake!").

Allora come siamo protetti? La sicurezza deriva dalla **Trasparenza** ("Security comes from transparency"):

1. Il certificato falso viene registrato nel Log Pubblico.

2. Il legittimo proprietario (Google) monitora il log, vede il certificato non autorizzato.

3. Il proprietario richiede la **Revoca** immediata del certificato alla CA.

4. Il browser dell'utente controlla lo stato di revoca (CRL/OCSP) e rifiuta la connessione.

In sintesi, mentre la Blockchain cerca di impedire che il male accada (validazione a priori), la Certificate Transparency garantisce che, se il male accade, sia visibile a tutti affinché si possa reagire (rilevamento a posteriori).

Analisi Didattica: Il Buttafuori vs La Telecamera

Per capire la differenza tra Blockchain e CT, usiamo un'analogia di sicurezza fisica:

- **Blockchain è come un Buttafuori:** Controlla il tuo invito all'ingresso. Se è falso, non entri. Il sistema impedisce preventivamente l'azione illegittima.
- **Certificate Transparency è come una Telecamera di Sorveglianza:** La telecamera registra chiunque entri, anche i ladri. Non ti impedisce di entrare, ma assicura che il tuo volto sia registrato in modo indelebile. Se un ladro entra (un certificato falso viene emesso), la polizia (il proprietario del dominio) può guardare il nastro, identificarlo e neutralizzarlo (revoca).