Protocolli basilari per la sicurezza

*Appunti di **Giuseppe Pitruzzella** - Corso di Internet Security @ DMI, UniCt*

Introduzione

Iniziamo ad operare il passaggio dalla crittografia alla sicurezza. Come si utilizzano in pratiche le primitive crittografiche studiate nel capitolo precedente? Lo vediamo all'interno di questo capitolo, in cui sposteremo il nostro focus sui protocolli di sicurezza, protocolli cui obiettivo è ottenere proprietà di sicurezza attraverso delle misure crittografiche. In questi protocolli faremo le assunzioni peggiori dal punto di vista offensivo, ovvero le assunzioni Dolev-Yao.

La **sintassi** utilizzata per discutere i protocolli in questo capitolo è la seguente:

- *Messaggi atomici*
  - **Nomi degli agenti**: A, B, C...
  - **Chiavi crittografiche**: $k_a, k_b, \dots, k_a^{-1}, k_b^{-1}, \dots, k_{ab}, k_{ac}, \dots$
  - **Nonce**: $N_a, N_b, \dots$
  - **Timestamp**: $T_a, T_b, \dots$
  - **Digest** (i.e. *hash*): $h(m), h(n), \dots$
  - **Etichette**: "$Trasferisci\ 100 €\ \ dal\ conto\ di\ \dots$"
- *Messaggi composti*
  - **Concatenazioni**, in cui ciascuno può essere un crittotesto: m, m^1;
  - **Crittotesti**, in cui il testo all'interno può essere concatenazione: m_k.

> Si può autenticare un messaggio concatenato del tipo $(m, m^1)_k$? Ricordiamo che autenticazione indica la dimostrazione dell'identità dell'interlocutore, che per estensione della definizione ci riferiamo all'autenticazione del messaggio, ovvero verificare che il messaggio è stato inviato proprio dall'autenticatore ("*Autenticare $Alice$ attraverso il messaggio m...*"). La risposta è no, poichè se così fosse allora chiunque potrebbe svolgere la concatenazione di un messaggio e compromettere l'autenticazione del messaggio.

Il modello di attaccante più comune indica che possiamo immaginare un modello di rete in cui vi è un attaccante che intercetta ogni possibile comunicazione e modifica quest'ultimi a suo piacere. L'unica cosa che l'attaccante non può fare e rompere la cifratura.
Storicamente, i due autori raccontavano il loro modello basandosi su due agenti 007, i quali fidandosi l'uno dell'altro e si trovandosi in due posti lontani, devono comunicare in sicurezza avendo l'intera rete come nemico.
Vent'anni dopo, si osserva che il contesto è cambiato, per cui si ipotizza che un singolo nodo della rete può avere capacità offensive pari al Dolev-Yao.

Come si scrive un protocollo di sicurezza? Fondamentalmente, il tutto si basa sulla seguente notazione: (i) il numero del passo, (ii) il mittente, (iii) una freccia, (iv) il ricevente, (v) due punti, (vi) il messaggio cirttografico; il tutto viene ripetuto per ogni singolo passo del protocollo.

$$NumeroPassoMittente \longrightarrow Ricevente : MessaggioCrittografico$$

> E' importante notare che nel momento in cui basiamo il tutto sull'attaccante DY, allora il messaggio crittografico perde di significato poiché egli stesso potrebbe essere modificarlo.

Protocolli basilari per la segretezza

Un protocollo di sicurezza più *basilare* che studiamo si basa sulla crittografia **simmetrica**, motivo per cui prevede un chiave condivisa tra i soli $Alice$ e Bob (unico prerequisito), attraverso cui $Alice$ si può autenticare con Bob (o viceversa). Questo protocollo è robusto, anche contro DY pur patendo qualche attacco (?).

- $Prerequisito\,1$: Esiste una chiave k_{ab} condivisa tra i soli $Alice$ e Bob;

$$1.\ A \longrightarrow B\ :\ m_{k_{ab}}$$

D'altra parte, esiste anche la variante con la crittografia **asimmetrica**, la quale prevede certi prerequisiti.

- $Prerequisito\,1$: Tutti abbiano una coppia di chiave;
- $Prerequisito\,2$: $Alice$ deve avere la garanzia che la chiave pubblica di Bob sia effettivamente di Bob.

$$1.\ A \longrightarrow B\ :\ m_{k_b}$$

Protocolli basilari per l'autenticazione

Un protocollo basilare per l'autenticazione nel caso **simmetrico** prevede l'introduzione di un assunzione, ovvero che il ricevente dell'autenticazione sappia a chi è associata la chiave di sessione, con chi condivide la chiave di sessione.

- $Prerequisito\,1$: Esiste una chiave k_{ab} condivisa tra i soli $Alice$ e Bob;
- $Prerequisito\,2$: Bob può verificare il $Prerequisito\,1$.
  Relativamente al protocollo stesso, segue quindi un *ecrypt* del messaggio per ogni messaggio inviato da $Alice$ a Bob (o viceversa).

$$1.\ A \longrightarrow B\ :\ m_{k_{ab}}$$

D'altra parte, esiste il caso **asimmetrico**, in cui assumono i seguenti prerequisiti:

- $Prerequisito\,1$: $Alice$ possiede una chiave privata $k_{a^{-1}}$ valida;
- $Prerequisito\,2$: Bob può verificare che la chiave pubblica di $Alice$ sia effettivamente di $Alice$ stessa.

Relativamente al protocollo stesso, segue che se $Alice$ vuole autenticarsi con Bob, allora egli esegue l'encrypt del messaggio attraverso la sua chiave privata, il che garantisce autenticazione.

$$1.\ A \longrightarrow B\ :\ m_{k_{a^{-1}}}$$

Combinare segretezza ed autenticazione

Combinare i protocolli basilari per l'autenticazione e la segretezza è possibile; per fare ciò, infatti, sarà sufficiente e necessaria l'unione dei prerequisiti necessari ad ognuna delle due versioni di protocolli.
Nel primo caso, basato su crittografia **simmetrica**, notiamo un protocollo analogo a ciò che abbiamo visto per l'autenticazione:

- $Prerequisito\,1$: Esiste una chiave k_{ab} condivisa tra i soli $Alice$ e Bob;
- $Prerequisito\,2$: Bob può verificare il $Prerequisito\,1$.
  Relativamente al protocollo stesso, segue quindi un *ecrypt* del messaggio per ogni messaggio inviato da $Alice$ a Bob (o viceversa).

$$1.\ A \longrightarrow B\ :\ m_{k_{ab}}$$

Nel caso **asimmetrico**, chiaramente, le assunzioni diventano 4. In particolare, notiamo che attraverso il c

- $Prerequisito\,1\ :\ Alice$ possiede una chiave privata $k_{a^{-1}}$ valida;
- $Prerequisito\,2\ :\ Bob$ può verificare che la chiave pubblica di $Alice$ sia effettivamente di $Alice$ stessa.
- $Prerequisito\,3\ :\ Bob$ possiede una chiave privata $k_{b^{-1}}$ valida;
- $Prerequisito\,4\ :\ Alice$ può verificare che la chiave pubblica di Bob sia effettivamente di Bob stesso.

Secondo quest'ultimo protocollo basato su crittografia **asimmetrica**, possiamo notare due approcci uguali a meno di qualche sfumatura.

- $1.\ A \longrightarrow B\ :\ \{m_{k_a^{-1}}\}_{k_b}$
  Attraverso questo primo approccio, certamente l'unico a poter decriptare il messaggio sarà il destinatario Bob, il quale potrà poi decriptare una seconda volta il messaggio interno secondo la chiave pubblica di $Alice$. Ne segue che questo approccio è migliore se l'obiettivo di $Alice$ è autenticarsi esclusivamente con Bob, quindi garantire la segretezza.
- $1.\ A \longrightarrow B\ :\ \{m_{k_b}\}_{k_a^{-1}}$
  Attraverso questo secondo approccio, certamente tutti possono decriptare il messaggio destinato a Bob; tuttavia rimane Bob l'unico che potrà poi decriptare una seconda volta il messaggio interno secondo la sua chiave privata. Ne segue che questo approccio non è il migliore per garantire la segretezza.

Protocolli basilare per l'integrità

Come abbiamo già detto, qualunque messaggio sulla rete potrebbe essere alterato, in particolare anche un messaggio protetto per segretezza ed autenticazione. A questo proposito, potrebbe essere utile l'approccio che prevede di affiancare ad ogni messaggio un checksum, come per esempio, l'hash del messaggio.
Tuttavia, fare ciò non nega all'attaccante che vuole sostituire m con m^1, di sostituire allo stesso $h(m)$ con $h(m^1)$.
E' necessario aggiungere un livello di irrobustezza.

Attraverso la crittografia **asimmetrica**, possiamo potenziare l'approccio precedente facendo uso di una chiave, ovvero la chiave privata del mittente $Alice$. In particolare, criptiamo l'hash del messaggio $h(m)$ con la chiave privata di $Alice$, ottenendo $h(m)_{k_{a^{-1}}}$.
In questo modo, anche se l'attaccante ricalcolasse l'hash egli non potrebbe mai eseguire l'encrypt di quest'ultimo.

$$1.\ A \longrightarrow B\ :\ m,\ h(m)_{k_{a^{-1}}}$$

A questo punto, quando Bob riceve il messaggio di $Alice$, allora egli (i) calcola l'hash del messaggio, (ii) decodifica $h(m)_{k_{a^{-1}}}$ per poi confrontare i due risultati. Se i due digest sono uguali, allora viene garantita l'integrità.
E' importante notare che Bob, con l'operazione (ii) riesce effettivamente ad individuare se il costruttore del digest è anche chi ha inviato il messaggio, ovvero $Alice$. Il tutto, quindi, garantisce autenticazione.

> Non è possibile garantire l'integrità senza garantire l'autenticazione.

La firma digitale

La **firma digitale** ha lo scopo di garantire l'integrità di un documento digitale ed è basata sulla crittografia asimmetrica. I requisiti funzionali di una firma digitale si basa sul fatto che ognuno possa applicare la sua firma digitale per conto di se stesso e mai per conto di altri; inoltre, ognuno deve poter verificare la

genuinità di quest'ultima firma (il che non è per una firma cartacea). Una firma digitale si basa su una coppia di **algoritmi**:

- Un algoritmo per la **creazione**;
  Questo primo algoritmo si basa sulle seguenti 3 fasi:
  1. Se devo firmare un documento $Cleartext$, allora inizialmente ne eseguo l'hash, costruendo il **digest**.
  2. In seguito eseguo l'encrypt del digest con la chiave provata del mittente (i.e. costruttore della firma), ottenendo il **digest crittografato** $Encrypted\,Digest$.
  3. A questo punto, la coppia $Cleartext + Encrypted\,Digest$ è uguale alla firma digitale.

  > Si noti che relativamente al concetto di firma digitale, non si è mai parlato di segretezza relativa al documento.

- Un algoritmo per la **verifica**.
  Ricevuta un documento firmato, ovvero una coppia $Cleartext + Encrypted\,Digest$, è possibile verificare la genuinità della firma secondo le seguenti 3 fasi:
  1. Eseguo il decrypt di $Encrypted\,Digest$ attraverso la chiave pubblica del mittente (i.e. creatore della firma), ottenendo il digest $Digest^1$.
  2. In seguito, effettuo l'hash del documento $Cleartext$, ottenendo il digest $Digest^2$.
  3. La firma è verificata se $Digest^1 = Digest^2$, il che indica che la coppia ha attraverso la rete senza alterazioni.

  > Qual è la differenza sul piano della sicurezza tra firma grafometrica e digitale?
  > Esistono due grandi differenze rispetto le due: infatti, per la prima diversamente dalla seconda, non esiste (i) autenticazione e (ii) integrità, il che indica che non può garantire che la firma sul foglio sia inalterato.

> Si noti che la chiave privata di cui abbiamo parlato all'interno della firma digitale è fornita da un fornitore di identità digitale.

Il protocollo Diffie-Hellmann

Il protocollo **Diffie-Hellmann**, pubblicato nel 1976, si basa sulla crittografia **simmetrica** (pur somigliando molto alla crittografia asimmetrica data la distinzione delle due chiavi) ed ha l'obiettivo di stabilire un segreto condiviso tra $Alice$ e Bob.
$Alice$ e Bob concordano due parametri **pubblici**, α e β. A questo punto, $Alice$ e Bob generano in modo random rispettivamente X_a ed X_b. A partire da quest'ultimi due, seguono:

$$Y_a = \alpha^{X_a}\ mod\ \beta$$

$$Y_b = \alpha^{X_b}\ mod\ \beta$$

.

> Perchè spedire Y_a sul traffico non espone X_a? Sappiamo che l'inverso dell'esponenziale sia il logaritmo, tuttavia se qualcuno stesse pensando che intercettando una tra le due Y sia possibile arrivare ad α svolgendo l'inverso (i.e. logaritmo), quest'ultimo si sbaglia.

> Infatti, aggiungere il modulo all'interno del calcolo di Y, rende l'operazione non più un semplice esponenziale, bensì un *logaritmo discreto*, un problema intrattabile.

Nel momento in cui $Alice$ e Bob scambiano le proprie Y, ricevendo rispettivamente Y_b ed Y_a, entrambi *elevano* rispetto il proprio segreto, calcolando: $Y_b^{X_a} \ mod \ \beta$ ed $\ Y_a^{X_b} \ mod \ \beta$. A questo punto, i due risultati saranno uguali ($Y_b^{X_a} \ mod \ \beta \ = \ Y_a^{X_b} \ mod \ \beta$), motivo per cui possiamo intendere quest'ultimo come il segreto condiviso per i due interlocutori.

$$A \ \ calcola \ \ k_{ab} = Y_b^{X_a} \ mod \ \beta \ = (\alpha^{X_b} \ mod \ \beta)^{X_a} \ mod \ \beta$$

$$B \ \ calcola \ \ k_{ab} = Y_a^{X_b} \ mod \ \beta \ = (\alpha^{X_a} \ mod \ \beta)^{X_b} \ mod \ \beta$$

$$k_{ab} = Y_b^{X_a} \ mod \ \beta \ = \ Y_a^{X_b} \ mod \ \beta$$

> E' importante ricordare che pur definendo le due chiavi X_a ed X_b come chiavi "*private*", queste non sono le chiavi private legate al concetto di crittografia asimmetrica ed il protocollo DH è definito secondo crittografia simmetrica.

Il protocollo Diffie-Hellmann, purtroppo, non ha previsto l'**autenticazione**, il che indica che $Alice$ e Bob si scambiano rispettivamente Y_a ed Y_b ma entrambi non hanno piena conoscenza di chi sia la persona con chi stanno per concordare una segreto.
Ha senso parlare di segretezza di un messaggio a meno dell'autenticazione? No, mai; infatti, supponendo che vi sia un segreto tra due persone, se quest'ultime non si autenticano allora il tutto ha poco senso.

Esiste un **attacco** relativo a DH ed all'assenza di autenticazione in quest'ultimo, ovvero:

1. $Alice$ invia a Bob il messaggio contenente Y_a.
2. Il messaggio viene intercettato da $Charlie$, il quale impersonando $Alice$, invia a Bob il messaggio Y_c.
3. Bob, in seguito invia ad $Alice$ il suo Y_b.
4. Il messaggio viene intercettato da $Charlie$, il quale impersonando Bob, invia ad $Alice$ il messaggio Y_c.

> All'interno del punto abbiamo scritto di Charlie che si impersona in Alice, inviando Y_c. Tuttavia, poiché assente l'autenticazione, è formalmente sbagliato scrivere di "*impersonificazione*".

E' importante notare che dato $k_a = Y_c^{X_a} \ mod \ \beta$ e $\ k_b = Y_c^{X_b} \ mod \ \beta$, conosciute rispettivamente da $Alice$ e Bob, allora $k_1 \neq k_2$.
Falliscono, quindi, (i) il **requisito funzionale** per cui $Alice$ e Bob potessero avere un segreto condiviso e (ii) la **segretezza** poiché l'attacante, avendo intercettato le due Y, può calcolare sia $k1 = Y_a^{X_c} \ mod \ \beta$ e $\ k2 = Y_b^{X_c} \ mod \ \beta$, motivo per cui si genera un attacco *man-in-the-middle*.

L'attacco, quindi, si divide in due fasi:

- Una prima fase in cui per $Charlie$ è possibile intercettare le due Y (le quali non sono segrete) e, data l'assenza di autenticazione, somministrare un nuovo messaggio ad $Alice$ e Bob.
- Una seconda fase in cui $Charlie$ può fare uso delle informazioni Y_a ed Y_b.

Esistono delle **modifiche** che permettono di irrobustire DH rispetto la prima e/o la seconda fase dell'attacco.
La prima fase può essere irrobustita facendo uso della chiave privata del mittente; in questo modo è

possibile inserire una misura di autenticazione. La seconda fase può essere irrobustita facendo uso della chiave pubblica del destinatario; in questo modo è possibile inserire una misura di segretezza.
Basta impedire una delle due fasi per fixare l'attacco a DH.
Purtroppo, il fix richiede di ricorrere alla crittografia asimmetrica. Un protocollo per la condivisione di un segreto autentico che non utilizzi primitive asimmetriche è un challenge aperto.

RSA Key Exchange

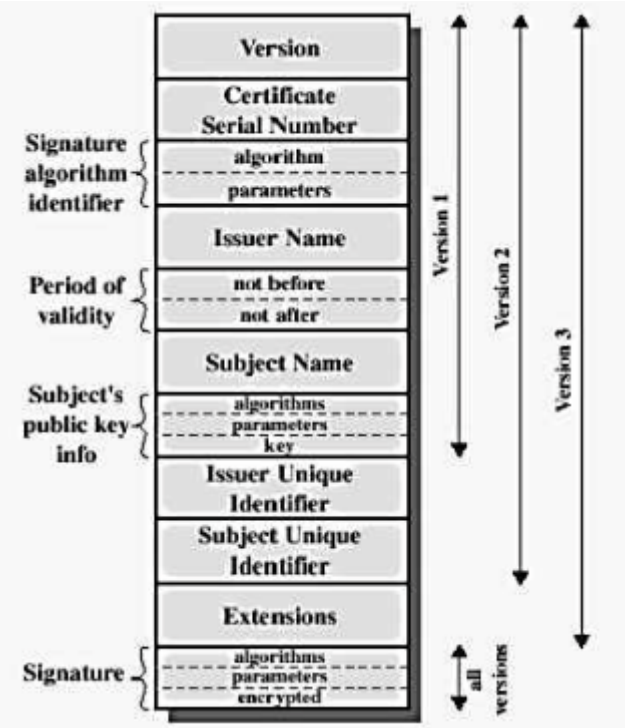
Un alternativa basata dichiaratamente nel mondo asimmetrico per lo scambio di una chiave esiste ed è **RSA**. RSA si basa sul seguente procedimento; il mittente, $Alice$, *genera randomicamente un segreto* ed esegue l'encrypt di quest'ultimo messsaggio con la chiave pubblica del destinatario. Questo è la scambio di una chiave secondo RSA.
Tuttavia, affinché il protocollo funzioni è necessario che Alice si procuri un **certificato** della chiave pubblica di Bob.

Certificazione

Un primo esempio di certificazione sappiamo essere la carta di identità, un documento che associa un informazione anagrafica ad un informazioni grafica. Tutti i certificati suggellano associazioni.

Un **certificato digitale** è l'associazione tra l'identità di $Alice$ e una stringa di bit, ovvero la sua *chiave pubblica*. Il certificato garantisce autenticazione ed integrità, tramite la chiave privata k_{ca}^{-1} di una autorità di certificazione;
Una versione semplificata di certificato sarebbe la seguente: $(Alice, k_A)_{k_{ca}^{-1}}$.
Tuttavia, al di là delle semplificazioni, il formato standard del certificato esiste e si chiama **X.509**.

Un certificato è firmato da un'autorità di certificazione (e.g. consorzi di banche), le quali si occupano principalmente di *vendere fiducia*, essere *trustworthiness*.
Si noti che i certificati hanno una validità temporale, ovvero hanno una scadenza.



Public Key Infrastructure (PKI)

Se $Alice$ e Bob vogliono eseguire una sessione *TLS* (basato su cifratura asimmetrica), allora $Alice$ necessita della chiave pubblica di Bob.

Il certificato è l'unico strumento che può dare questa informazione ad $Alice$.
Un certificato della forma *X.509* viene quindi inviato ad $Alice$, che **verifica** quest'ultimo, trovando nome e chiave pubblica di Bob.
Il certificato, per ovvi motivi, può essere alterato ma in quel caso la verifica fallisce.

Cosa vuol dire **verificare un certificato**? Verificare un certificato indica *verificare una firma digitale*, il che indica eseguire la seguente procedura di cui abbiamo già scritto nel capitolo relativo alla firma stessa.

1. Eseguo il decrypt di $Encrypted\,Digest$ attraverso la *chiave pubblica del creatore della firma*, ottenendo il digest $Digest^1$.
2. In seguito, effettuo l'hash del documento $Cleartext$, ottenendo il digest $Digest^2$.
3. La firma è verificata se $Digest^1 = Digest^2$, il che indica che la coppia ha attraverso la rete senza alterazioni.

Per portare a compimento la procedura di verifica è necessaria la conoscenza della chiave pubblica di ca, ovvero k_{ca}, il che ci porta nuovamente al problema iniziale, ovvero: "*chi mi assicura che quella chiave pubblica sia effettivamente dell'ente certificante*?".
E' necessario quindi *re-iterare* il problema (i.e. *verificare una catena di fiducia*) finché non sarà necessario un atto di fede nei confronti dell'**autorità root**.

La catena formata a partire dalla *re-iterazione* del problema non è in realtà una catena, bensì un **albero *n-ario***, formato da una radice che certifica i suoi n figli, i quali a sua volta certificano i propri n figli e così via. Possiamo definire quest'ultimo albero n-ario come la **Public Key Infrastructure** (PKI).

In definitiva, ripetiamo che il problema della certificazione rispetto la chiave pubblica di Bob prevede una re-iterazione del problema, la quale termina nel momento in cui arriviamo all'autorità root. Si noti che il certificato di root possiede una particolarità rispetto le successive certificazioni date dalle autorità figlie; possiede, infatti, *al di fuori* la chiave privata di root e dentro la sua stessa chiave pubblica.

Nella pratica, è possibile notare tutto ciò di cui abbiamo scritto all'interno del nostro browser, ogni qual volta digitiamo un URL. Modifiche visive all'interno di quest'ultimo (i.e. URL) avvisano l'utente rispetto la genunità del sito Web.
Si noti che è quindi il browser ha decidere di chi fidarsi e di chi non fidarsi (e.g. Google con Semantyc) motivo per cui è necessario tener d'occhio i certificati di root relativi al browser stesso. Ciò è importante poiché pur tenendo conto dei certificati, rimaniamo attaccabili sotto questo punto di vista, per esempio attraverso l'aggiunta di un certificato di root avvenuta a partire da uno script malevolo.

Segretezza vs Trustworthiness

E' interessante chiedersi cosa succederebbe se, per ipotesi, un nodo all'interno dell'albero sia compromesso. In quel caso è importante notare che il danno che quest'ultimo nodo può provocare si muovo da egli stesso verso le foglie (i.e. *verso il basso*), non viceversa. Il tutto *non è retroattivo*, motivo per cui i certificati da attenzionare in tal caso saranno al più successivi all'evento che ha dato inizio alla compromissione del nodo (il che indica l'importanza dell'aspetto temporale).
Tuttavia, se un nodo è compromesso non necessariamente lo sono tutti i nodi al di sotto.
Non confondiamo tutto questo con la perdita di segretezza, la quale non si propaga, ma si riferisce esclusivamente alla perdita di trustworthiness.

Certificati nel browser

Sappiamo che vi è un business legato alle certificazioni, il che indica comprare un certificato per il proprio dominio ed eventualmente per i propri sotto-domini (leggermente diverso per certificati wild-card, quali permettono la certificazione anche per i propri sotto-domini)
Tuttavia, se non volessi comprare un certificato, quindi non volessi acquistare fiducia da qualcuno, allora l'unica opzione è crearsi un certificato da se, un certificato **self-issued**.

Supponendo di trovarsi in una situazione di questo genere, rappresentato dall'immagine sottostante. Esistono tre opzioni:

- Accettare il certificato in modo permanente;
- Accettare il certificato in modo temporaneo;
- Rifiutare il certificato e non connettersi al sito Web;

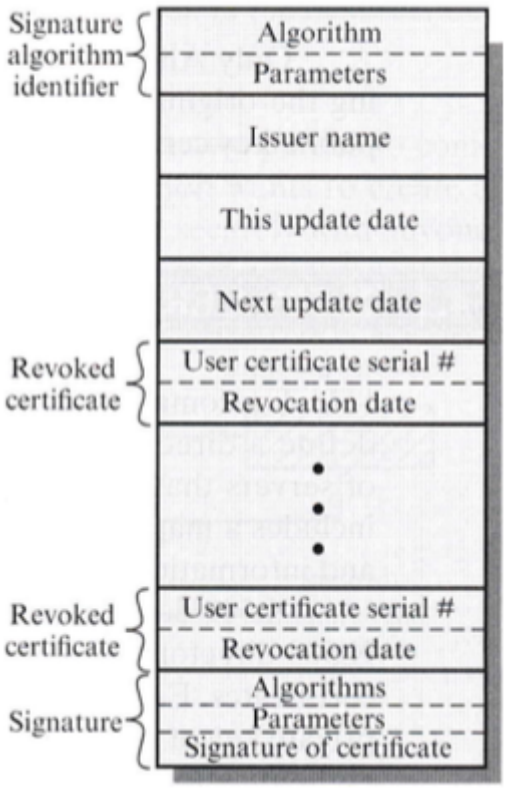
Qual è la scelta più sicura? Certamente se il browser potesse fare la scelta più sicura, allora sarebbe stato il browser stesso ad effettuare la scelta. In questo caso, siamo noi stessi a dover rispondere ad una domanda, una domanda a cui il browser non è in grado di rispondere.
L'utente potrebbe fare un verifica *out-of-band* (i.e. *fuori dal canale*), quindi (i) ispezionare il certificato, (ii) leggere l'hash (i.e. *certificate-finger-print*) e (iii) telefonare al centro di calcolo per verificare quest'ultimo hash.
Il tutto non è scalabile o usabile, motivo per cui potrebbe non accadere.
Rispondiamo, quindi, alla domanda. Accettare il certificato per sempre, implica che potrei potenzialmente essere fregato ogni volta da quel momento in poi; d'altra parte accettare il certificato in modo temporaneo implica accettare dei rischi limitati alla sessione, quindi essere fregati al più una sola volta.

> Negli ultimi tempi, un consorzio si è unito per mettere a disposizione una versione di certificato gratuito, questo è *Let's Encrypt*.

> Abbiamo capito che il senso della certificazione si basa sull'affermare che il contenuto è associato all'URL. Niente ci garantisce che un sito certificato non abbia scopi malevoli.

Revoca di un certificato, CRL

Relativamente al concetto di revoca esiste la **CRL** (acronimo di *Certificate Revocation List*), una lista di certificati revocati con la seguente struttura:



Un certificato revocato indica la revoca della validità del certificato prima della scadenza effettiva di quest'ultimo ed è conseguenza di *data-breach* o richiesta dell'interessato.

Un certificato revocato è firmato dalla medesima autorita di certificazione che ha firmato in principio il certificato stesso.
E' sempre necessario che la revoca di un certificato arrivi ad ogni foglia. In questo modo, evitiamo un **attacco** secondo cui l'attaccante, sfruttando il fatto non sia arrivata quest'ultima revoca, somministra all'utente qualsivoglia contenuto.
Tuttavia, purtroppo, non vi è alcuno standard (i.e. protocollo) rispetto la gestione della revoca. Per questo motivo, originariamente era l'utente stesso ad inserire le CRL all'interno del proprio browser. Ad un certo tempo nasce OCSP, attraverso cui è possibile sincronizzare il proprio browser rispetto una CRL.

> Il più recente strumento per le CRL è Certificate Transparency di Google. "*Per poter criptare il traffico per gli utenti, un sito deve innanzitutto richiedere un certificato a un'autorità di certificazione (CA) attendibile. Il certificato viene poi presentato al browser per l'autenticazione del sito a cui l'utente desidera accedere. Negli ultimi anni, le CA e i certificati emessi si sono rivelati vulnerabili alla compromissione e alla manipolazione, a causa di falle strutturali nel sistema dei certificati HTTPS. Certificate Transparency di Google è stato ideato per proteggere il processo di emissione dei certificati offrendo un framework aperto per il monitoraggio e il controllo dei certificati HTTPS. Grazie al log Certificate Transparency attivi e pubblici, i proprietari dei siti possono cercare su questo sito i nomi di dominio da loro gestiti per verificare che non ci siano errori di emissione dei certificati che fanno riferimento ai loro domini. Google invita tutte le CA a scrivere i certificati emessi in log a prova di manomissione, di tipo append-only e pubblicamente verificabili.*".