Abstract

Questo è un documento in cui si collezionano gli appunti sui *principid i Cybersecurity* tratto dal corso *Reti di Calcolatori e Principi di Cybersecurity*, che fa parte del programma della seconda parte del corso (non è tutta la seconda parte!)

Si sconsiglia quindi di usare questo materiale in alternativa alle lezioni o alle slide fornite dal docente A. Bartoli, in quanto i miei appunti potrebbero contenere refusi, errori di battitura, oppure potrebbero essere dificili da comprendere senza aver avuto prima un'idea dei contenuti, che sono ottenibili seguendo le lezioni.

Detto ciò, non mi assumo nessuna responsabilità del vostro rendimento in questo corso se decidete di basarvi su questi appunti.

"Network Security"

Introduzione alla Network Security

X

Introduzione alla Network Security (≈ Cybersecurity): definizioni preliminari di subject e subject fisico, problema caratterizzante della Network Security. Esempi motivanti: HTTP injection, DNS spoofing. Scaletta del capitolo.

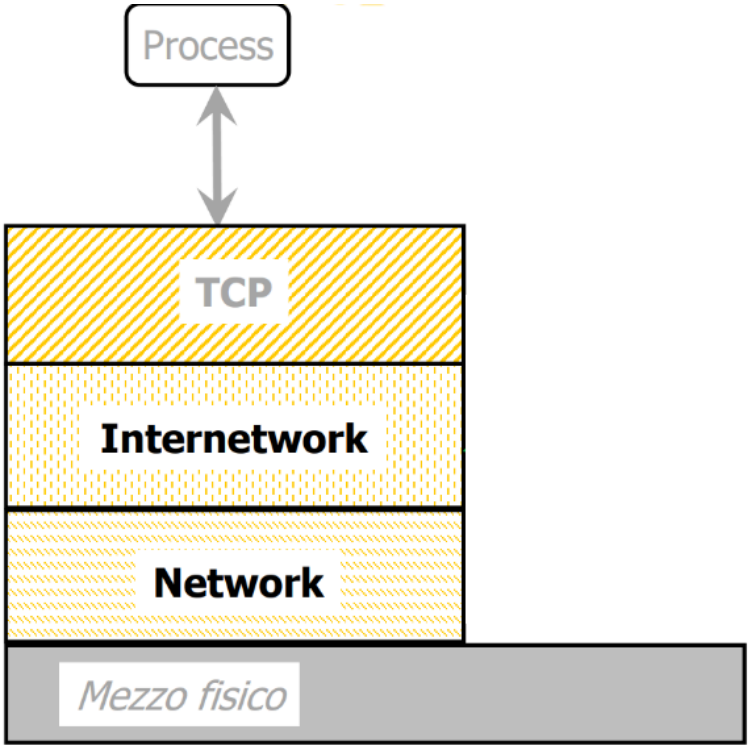
X

0. Voci correlate

- Comunicazione tra Processi
- Protocollo HTTP
- Aspetti Strategici del DNS

1. Network Security: Definizioni

Adesso torniamo al *livello applicativo* dello stack dei protocolli di Internet.



In particolare, approfondiremo su come possiamo applicare le *tecniche di crittografia* su applicazioni *quotidiane*.

Diamo prima delle definizioni preliminari:

B. Con B si denota una *sequenza di bytes* "qualsiasi", che può rappresentare:

- Un messaggio di un protocollo, come SMTP, POP, HTTP, IP/PPP, o anche dei *frame*
- Un file, una email, ...

Mittente/Creatore/Proprietario. Quando una sequenza di bytes B viene trasportata, si ha l'idea del *"mittente"* di B, o similmente l'idea del *"creatore"*/*"proprietario"* (dipende dal contesto). Si definiscono in due modi:

- *Esplicitamente*: la si specifica in un certo punto di B (ex: mail)
- *Implicitamente*: non viene specificato (ex: richiesta HTTP di una sessione autenticata)

Osservazione. Per *"mittente/creatore/proprietario"* si intendono anche i *processi* che creano B, non solo il lato utente

Subject e Subject Fisico. Un *subject* è una *stringa* nel calcolatore che indica l'utente (è circa l'*username*). Ad ogni subject si associa un *subject fisico*, che è un'utente o un'organizzazione esterna al calcolatore. Spesso ogni subject fisico è associato a più subject.

Esempio. (*Subject e Subject Fisico*)
Il DNS name è un esempio di *subject*, con subject fisico il *registrant* del DNS name.

Canale di Comunicazione. Quando un subject S genera B e la vuole trasportare ad un altro subject S', la si fa mediante un *canale di comunicazione*. Notiamo che oltre a rappresentare uno spostamento nello spazio, si rappresenta anche uno spostamento nel tempo.

Osservazione. Il canale di comunicazione la si intende nel *senso astratto*, ovvero non è necessariamente Internet o la network. Può essere generalizzata anche in altri ambiti! (esempio: GPS)

X

2. Problema Fondamentale della Network Security

Vediamo il *problema unico* che caratterizza il campo della *network security*.

Problema.

Nel canale di comunicazione esiste un *network attacker* NA che vuole *"impedire"* il trasporto di B da S a S'. In particolare, supponiamo che NA sia in grado *osservare, modificare e*

> *fabbricare* una qualsiasi sequenza di bytes che *"scorre"* nel canale di comunicazione.

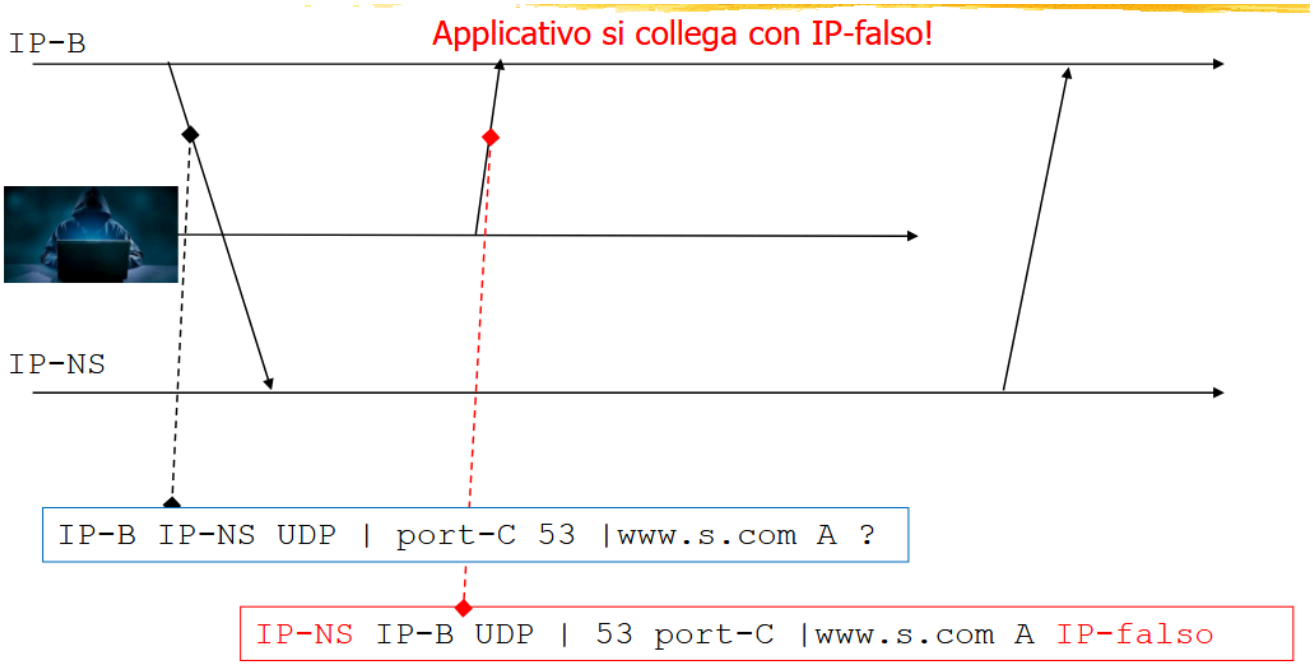
Pertanto diventa impossibile comunicare a tutti gli effetti; uno degli obbiettivi della *Network Security* è quello di definire delle *proprietà di sicurezza* sulla sequenza di bytes B e vedere come implementarla.

Osservazione. Si assume che la presenza del Network Attacker sia un *dato di fatto* a priori, siccome vogliamo pensare solo alle *conseguenze* del problema. Non pensiamo a *come sia possibile*, ed è un altro problema che è interamente diverso da quello che vogliamo risolvere.

Vediamo un paio di esempi motivanti, per sottolineare quali siano le *conseguenze* del problema posto sopra (^9d00d1)

Esempio. (*HTTP injection*)
Supponiamo che il browser stia comunicando con un Web Server, e preleva la pagina dell'home page mandando una richiesta HTTP. Quello che può fare il network attacker *"in mezzo"* è quello di modificare la risposta HTTP modificando vari link (come quella per la pagina di login), così quando l'utente invia una richiesta POST per inviare le credenziali, lo manda al nodo del *network attacker*, non del *web server*.

Esempio. (*DNS spoofing*)
Quando un nodo invia una DNS request, è possibile che il network attacker lo intercetti e invia una risposta DNS *"fasulla"* più velocemente del *"DNS vero"*.



Esempio. (*Network Attacker*)
Ci sono molti casi in cui può esistere il network attacker, come:

- Sistemisti Access Point/Router/Name Server/ISP
- Enti statali/giudiziarie che impongono sorveglianza a ISP
- Reti WiFi aperte, *"tutti"* sono potenzialmente dei network attacker
- Reti WiFi *"personal"* in ambienti *"non personal"*
    - *Osservazione. Nelle reti WiFi, distinguiamo le reti "personal" da "enterprise" a seconda di quanto gli utenti autenticati si "fidino tra di loro".*

X

---

3. Scaletta

Quindi in questo capitolo vedremo tre aspetti della Network Security:

1. Definire le proprietà di sicurezza richieste
2. Definire dei strumenti matematici e studiarli dal punto di vista funzionale ("come si usano?")
3. Usare 2. per realizzare 1.

Proprietà di Sicurezza

X

Proprietà di sicurezza. Authentication, Riservatezza ed Integrità. Esempi. Osservazioni: nessun protocollo viste fin'ora garantisce queste proprietà, authentication e riservatezza da un punto di vista pratico. Altre proprietà importanti. Osservazione: le proprietà di sicurezza "esistevano" prima dell'Internet.

X

0. Voci correlate

- Introduzione alla Network Security

1. Le Tre Proprietà Fondamentali

Authentication

Supponiamo che in una sequenza di bytes B ci sia informazione *implicita/esplicita* sul suo *"mittente/proprietario"* subject S. *Authentication* consiste nel garantire che B sia *"veramente"* generata da S

Esempio. Richieste e risposte HTTP. Quando si garantisce che una richiesta HTTP sia generata *veramente* da un certo browser? Quando si garantisce che la *risposta HTTP* (con la pagina) sia veramente generata dal web server?

Esempio. Possiamo vedere i protocolli su livelli più bassi, come:

- Frame Ethernet con ETH-src = ETH-x: come garantiamo che sia veramente da ETH-x?
- Pacchetto IP con IP-src = IP-x: come garantiamo che sia veramente dal nodo di IP-x?

Esempio. Anche applicabile su *bytes intesi come dati*, vedere ad esempio il caso della *mail address spoofing* (Email Spam e Address Spoofing)

Riservatezza

Sia B una *sequenza di bytes* inviata da S a S' tramite un canale di comunicazione. La *riservatezza* è quando B è *"comprensibile"* solo ad un insieme di *"subject autorizzati"*

Vedremo di capire meglio le nozioni di *"comprensibile"* e *"subject autorizzati"* in seguito.

Esempio. Richieste HTTP con Username e Password... importante che siano comprensibili solo al WS!

Naturalmente ci sono altri numerosissimi esempi applicati a tipologie di B diverse, come files, immagini, traffico DNS con nomi dei siti, eccetera...

Integrità

Sia B costruita all'istante di tempo t_1 e *"usata"* all'istante di tempo t_2. *Integrity* consiste nel garantire che B sia sempre lo stesso nelle due istanti di tempo, ossia

$$B(t_1) = B(t_2)$$

Osservazione. Luogo di costruzione può essere $\neq$ al luogo di verifica, come il subject che costruisce B può essere $\neq$ al subject che verifica

Esempio. Pagina web contenente *link critico* (login page):

- t_1: Invio dal web server
- t_2: Ricezione dal browser

Esempio. Referto medico

- t_1: Costruzione del referto
- t_2: Analisi in procedimento giudiziario (eventuale)

In alcuni casi l'*integrità* diventa un'aspetto cruciale.

Notiamo che nessun protocollo tra quelli visti fin'ora garantite le proprietà appena definite, al massimo abbiamo solo *autenticazione al lato client* in situazione *"senza network attacker"*.

Osservazione. Nelle applicazione pratiche vogliamo garantire l'*autenticazione e la riservatezza* anche al livello di *subject fisico*, ovvero vogliamo garantire che un *subject* sia veramente associato al *subject fisico*. Questo richiede dei procedimenti aggiuntivi che non vedremo

Osservazione. Integrity non ha nulla a che fare con Subject

1.1. Altre proprietà

Ci sono delle proprietà altrettanto *importanti* che non vedremo in profondità:

- *"Non Repudio"* e *"Plausible Deniability"*: Garantire che un subject non può negare di aver mandato B, o che può dimostrare di non aver mandato B
- *"Timestamp"*: B deve esistere in un istante specificato
- *Availability*: Il sistema funziona, ovvero il network attacker non blocca la comunicazione (!)
- ...

Osservazione. Tutte le proprietà viste sono sempre state richieste sin dai tempi dei *romani*. Ciò implica naturalmente che ci sono state delle soluzioni *"pre-Internet"* per garantire le suddette proprietà, e tipicamente si basavano su due elementi:

- Contatto fisico tra subject fisici
- Comunicazioni o documenti cartacei

Crittografia a Chiave Privata e HMAC

X

Strumenti matematici per la Network Security. Premessa: assunzione della perfezione. Crittografia a chiave privata: idea, algoritmo e correttezza, esempio storico Enigma. HMAC: Idea, osservazioni e correttezza.

X

0. Voci correlate

- Proprietà di Sicurezza
- Introduzione alla Network Security

1. Premessa

PREMESSA. D'ora in poi assumiamo la *perfezione*:

- In implementazione degli algoritmi e protocollo
- Degli software
- Procedure operative

Questa è un'ipotesi piuttosto irrealistica, infatti spesso non vengono verificate e quindi crollano le proprietà di sicurezza per questo motivo.

Esempio. Per dare un'idea, abbiamo il seguente scenario:

- Per qualche errore sul software (vulnerabilità RCE, ad esempio) è in grado di eseguire dei programmi sul nodo
- Quindi il Network Attacker diventa Node Attacker
- Crollano le proprietà di sicurezza, siccome tengono solo del *Network Attacker*

Osservazione. Dunque gli strumenti matematici che vedremo non risolvono automaticamente i problemi! Infatti, per citare l'informatico britannico più importante nell'ambito della Cybersecurity,

> *"Whenever anyone says that a problem is easily solved by cryptography, it shows that he doesn't understand it"."*

X

2. Crittografia a Chiave Privata

Introduciamo il *primo strumento matematico*, la crittografia a chiave privata.

IDEA. Assumiamo che *due subject* condividano un *numero naturale* K di N bit, detta *"chiave privata"*. Utilizzano K per garantire la *secrecy*, in particolare usando algoritmi di *codifica e decodifica* parametrizzate in K che siano *"mutualmente inversibili"* per stesso parametro.

Notazione. Il procedimento di *"criptaggio di un messaggio con chiave K"* si denota con $\texttt{ENCRYPT}_K(m)$, invece il procedimento di *"decrittaggio di un messaggio cifrato con chiave K"* si denota con $\texttt{DECRYPT}_K(c)$.

Quindi formalmente, l'idea di base è quella di usare *uno/due* algoritmi di crittaggio e decrittaggio tali che

$$m = \texttt{DECRYPT}_{K'}\left(\underbrace{\texttt{ENCRYPT}_K(m)}_{c}\right) \iff K = K'$$

Adesso vediamo di fare un paio di precisazioni formale sulle *chiavi* K:

- Le chiavi sono sequenze di N bit
- Pertanto ci sono 2^N chiavi possibili per N fissato. Il valore N dipende dagli algoritmi usati...

Proprietà. Affinchè l'algoritmo possa garantire la *segretezza/riservatezza*, deve valere la seguente proprietà fondamentale:

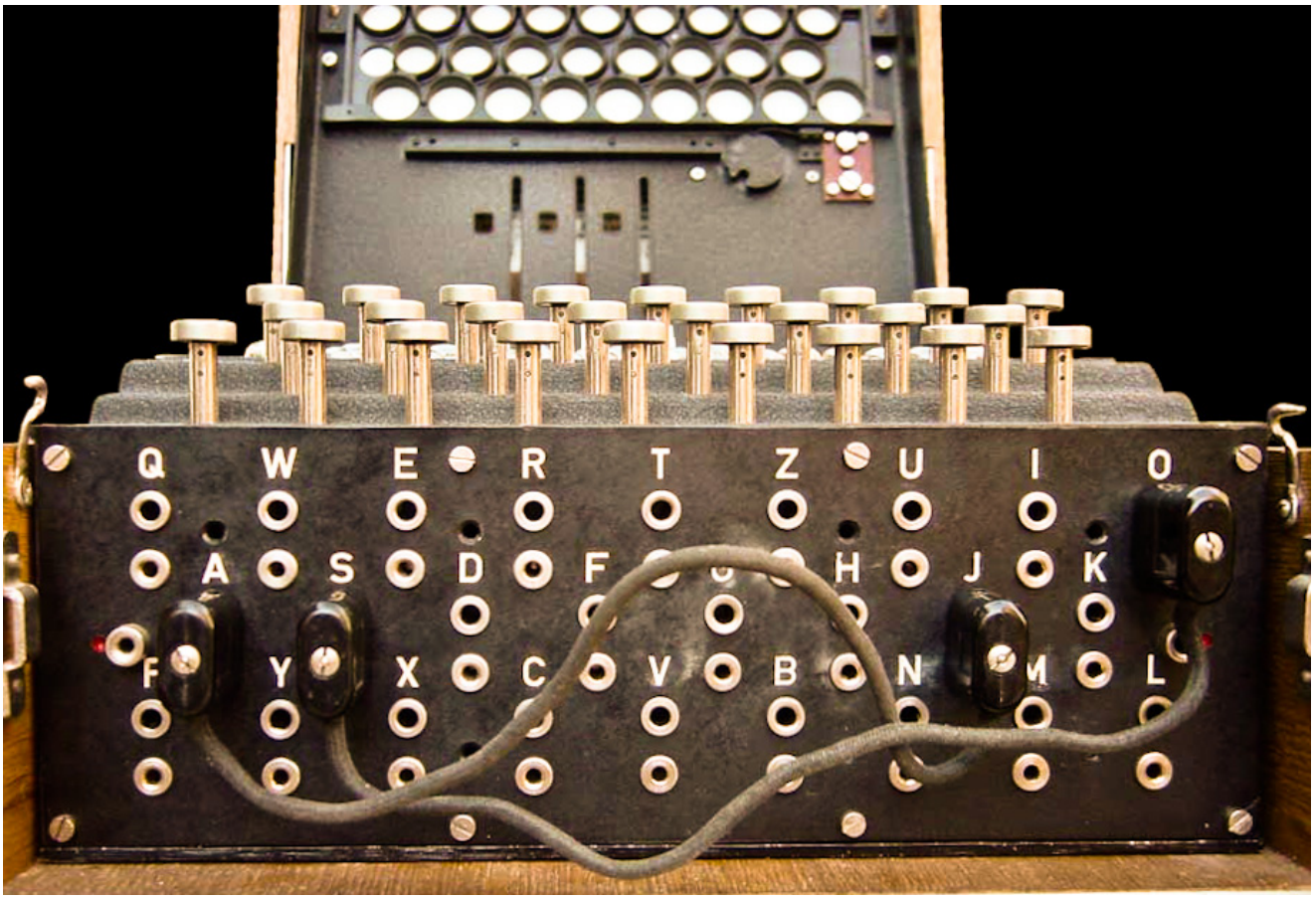
- E' *"difficile risalire"* a k da un testo cifrato
- E' *"difficile risalire"* al messaggio chiaro dal testo crifrato

In questo modo, un *network attacker* che conosce l'algoritmo di crittaggio/decrittaggio e che ha il testo cifrato, può tentare di evincere il plaintext in due modi:

1. *Ricerca esaustiva*: Indovinare la chiave K, quindi su 2^K chiavi possibili usare l'algoritmo $\texttt{DECRYPT}_K(\bullet)$ per evincere K e m (lo capisce quando il testo decrittato *"ha senso"*)
2. *Crittoanalisi"*: Osservare il testo cifrato ed evincere le informazioni sul chiave, in particolare informazioni di tipo *"di esclusione"* (nel senso che si esclude un sottoinsieme dello spazio delle chiavi), poi per applicare nuovamente la ricerca esaustiva sullo spazio ridotto

Tuttavia si dimostra che oggi entrambi gli approcci sono inutilizzabili per gli *algoritmi moderni*, siccome permettono N sufficientemente grande e non permette l'estrazione delle informazioni su K o plaintext.

Esempio. Un esempio storico della crittografia a chiave privata è la macchina *Enigma*, che però era vulnerabile alla *crittoanalisi* e infatti ad un certo punto della 2GM (seconda guerra mondiale) le chiavi private sono state scoperte, permettendo alle forze alleate di ottenere informazioni preziose.



Esempio. Alcuni algoritmi moderni:

- DES (56 bit, sono *"pochi"*)
- AES (128 o 256 bit), la più diffusa oggi
- ...

Osservazione. Le tecniche di crittografia non garantiscono comunque l'*integrità* dei messaggi... infatti, il Network Attacker può comunque modificare il *cypther text*, cambiando quindi il testo decrittato senza che il ricevente ne accorga. Analogamente, non si garantisce neanche l'*autenticazione*, dal momento che può inviare un *"messaggio casuale"* e mandarlo al ricevente.

X

---

3. HMAC (Message Authentication Code)

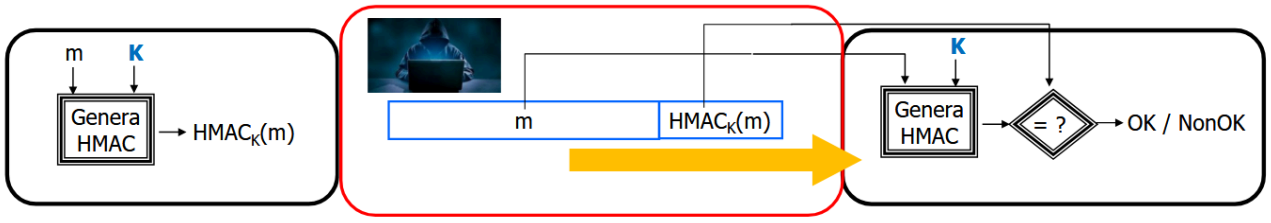
Vediamo un altro strumento per garantire un'altra proprietà di security.

Idea. Come sempre, S, S' condividono una chiave segreta K per garantire l'*autenticazione* e la *segretezza*. Si definisce una funzione parametrizzata in K, che denoteremo con $\texttt{HMAC}_K(\bullet)$, ed essa genera a partire da B un'altra sequenza di bytes con *lunghezza fissa*.

Per garantire le due proprietà sopra, si appende al messaggio originale m la nuova sequenza generata, ovvero $m' = m \uplus \texttt{HMAC}_K(m)$. In questo modo, quanto S' riceve $(m \uplus \texttt{HMAC}_K(m))$, può prima calcolare $\texttt{HMAC}_{K'}(m)$, e si ha che

$$\texttt{HMAC}_{K'}(m) = \texttt{HMAC}_K(m) \iff K = K'$$

Quindi il messaggio è *autentico e integro* siccome chi ha prodotto il messaggio conosce la chiave privata, pertanto il lato ricevente dovrà solo controllare che i valori HMAC coincidano.



Osservazione. In ogni caso, *HMAC* da sola non fornisce la *secrecy*! Quindi un network attacker può vedere il messaggio in chiaro, ma non può nè falsificare nè modificare il messaggio senza che il receiver che se ne accorga.

Si dimostra che gli algoritmo odierni di HMAC sono tali che ottenere $K, \texttt{HMAC}_K(m)$ solo da m sono *"praticamente impossibili"* (nel senso che l'attacker può solo usare la forza bruta).

Osservazione. Con l'HMAC leghiamo la nozione di *autenticazione* alla *conoscenza* della chiave K, quindi del software che conosce quel valore, invece non c'è nessun cenno di *subject* o *subject fisico*. Infatti, come ipotesi implicita si assume che *solo* S, S' conoscano K.

X

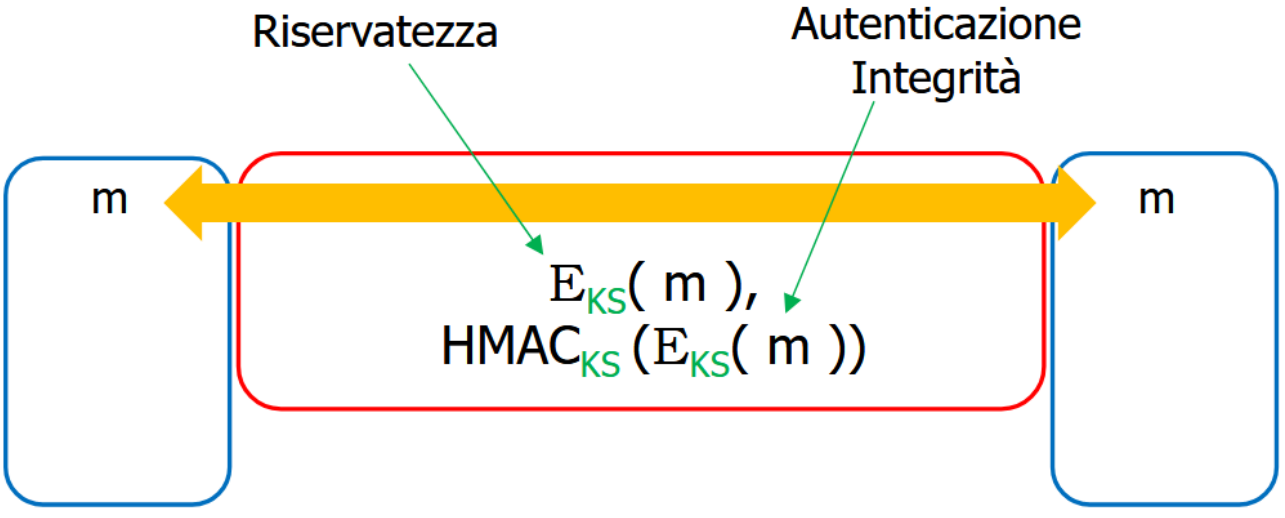
---

3. Realizzazione Parziale delle Proprietà di Sicurezza

Osservazione. Notiamo che sotto l'assunzione di aver distribuito correttamente le *chiavi segrete* tra due subject, il gioco è stato fatto. Infatti, per garantire tutte le proprietà di comunicazione (segretezza, autenticazione e integrity) si modifica il messaggio m nel seguente modo:

$$m \leftarrow \texttt{ENCRYPT}_K(m) + \texttt{HMAC}_K(\texttt{ENCRYPT}_K(m))$$

Nella matematica della crittografia si dimostra che è necessario *prima* crittare il messaggio e dopo calcolare l'*HMAC*, altrimenti si rischia di fare un *"leak"* sulla chiave o messaggio in chiaro.



Quindi tutto è stato fatto... o no? (*Spoiler: No, vedere l'assunzione di base degli strumenti matematici*)

Crittografia a Chiave Pubblica

X

Motivazione: distribuzione delle chiavi private. Crittografia a chiave pubblica: idea dell'algoritmo, proprietà fondamentale. Esempio: RSA. Osservazione: novità "storica" del strumento matematico, curiosità storiche. Osservazione: differenza dalla crittografia a chiave privata, non garantisce né integrità né autenticazione. Terminologia per crittografia a chiave privata e pubblica: asimmetrica e simmetrica.

X

0. Voci correlate

- Introduzione alla Network Security
- Proprietà di Sicurezza
- Crittografia a Chiave Privata e HMAC

1. Distribuzione delle Chiavi Private

Osservazione. La crittografia a chiave privata e HMAC è in grado di garantire le proprietà di sicurezza se c'è un'ipotesi fondamentale di base: ovvero, che le *chiavi private* vengono condivise tra Subject.

Q. Come fanno i subject a condividere la chiave privata?

Il problema della *distribuzione delle chiavi private* è un altro grande problema.

Esempio. (*Storico*)
Storicamente, si usava un altro *canale sicuro* di comunicazione che per definizione garantisce le proprietà di comunicazione. Ad esempio, incontrarsi di persona e scambiare le chiavi con valigette diplomatiche o oggetti del genere.

Problema. Il canale sicuro è *costoso* e *"ad alta latenza"*, quindi veniva utilizzato *"poco"* in storia. Tuttavia, il problema è diversamente adesso: ci sono *tantissimi coppie di Subject* che

vorrebbero scambiarsi chiavi private, basta pensare ad esempio ogni utente di GMAIL.

Infatti, se il canale sicuro fosse economico e a "bassa latenza" non ci sarebbe stata nessuna necessità di designare le *chiavi private e HMAC*... basta utilizzarli direttamente

In pratica, il canale aggiuntivo è utilizzabile solo in casistiche molto specifiche:

- Poche coppie di Subject
- Regole note a priori
- Staticità

Che è il contrario dello scenario *"Internet"*.

Come si fa? Vedremo uno strumento matematico in più...

Osservazione. La crittografia quantistica NON RISOLVE QUESTO PROBLEMA! Infatti, parte sempre dal presupposto che le chiavi siano già distribuite...

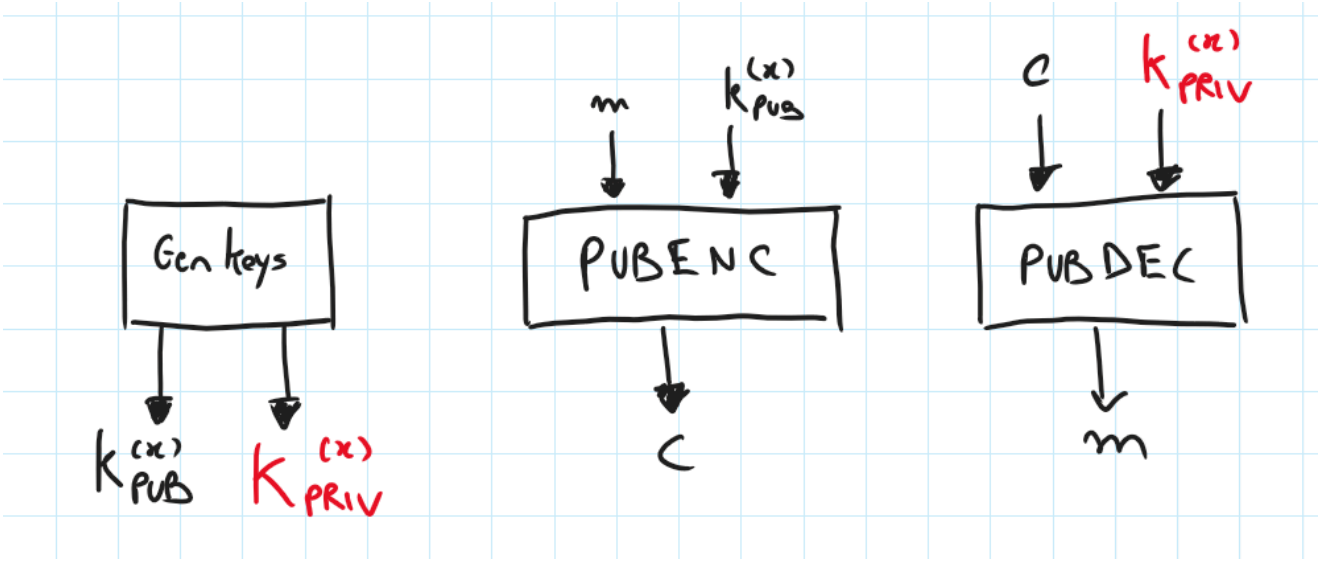
X

2. Crittografia a Chiave Pubblica

IDEA. Un algoritmo *genera* una coppia di chiavi $(K_{\texttt{PRIV}}, K_{\texttt{PUB}})$ chiamati *"chiave privata"* e *"chiave pubblica"* (vedremo perché dopo) che abbiano una *"relazione matematica"* tra loro, poi altri *due algoritmi* di crittaggio e decrittaggio pubblico, scritti come $\texttt{PUBENCRYPT}_\bullet(\bullet)$ e $\texttt{PUBDECRYPT}_\bullet(\bullet)$ soddisfano la seguente equivalenza:

$$\texttt{PUBDECRYPT}_{K'_{\texttt{PRIV}}}(\texttt{PUBENCRYPT}_{K_{\texttt{PUB}}}(m)) = m \iff K'_{\texttt{PRIV}} = K_{\texttt{PRIV}}$$

In parole povere, solo chi ha la chiave privata può decrittare il messaggio ma tutti possono crittare il messaggio.



Osservazione. Lo spazio delle chiavi *sono un sottoinsieme* delle coppie dei numeri a N bit, siccome devono soddisfare delle relazioni matematiche.

Le proprietà fondamentali che garantiscono delle proprietà di sicurezza sono come seguono:

- Dalla chiave pubblica non *"si evince"* la chiave privata
- Dal messaggio cifrato non *"si evince"* la chiave privata
- Dal messaggio cifrato non *"si evince"* il messaggio in chiaro

Questo strumento è in grado di realizzare la *segretezza* se partiamo dal presupposto che la chiave pubblica sia *"già distribuita"* al nodo sorgente. Infatti, l'attacker che conosce l'algoritmo e chiave pubblica non può fare niente senza la chiave privata!

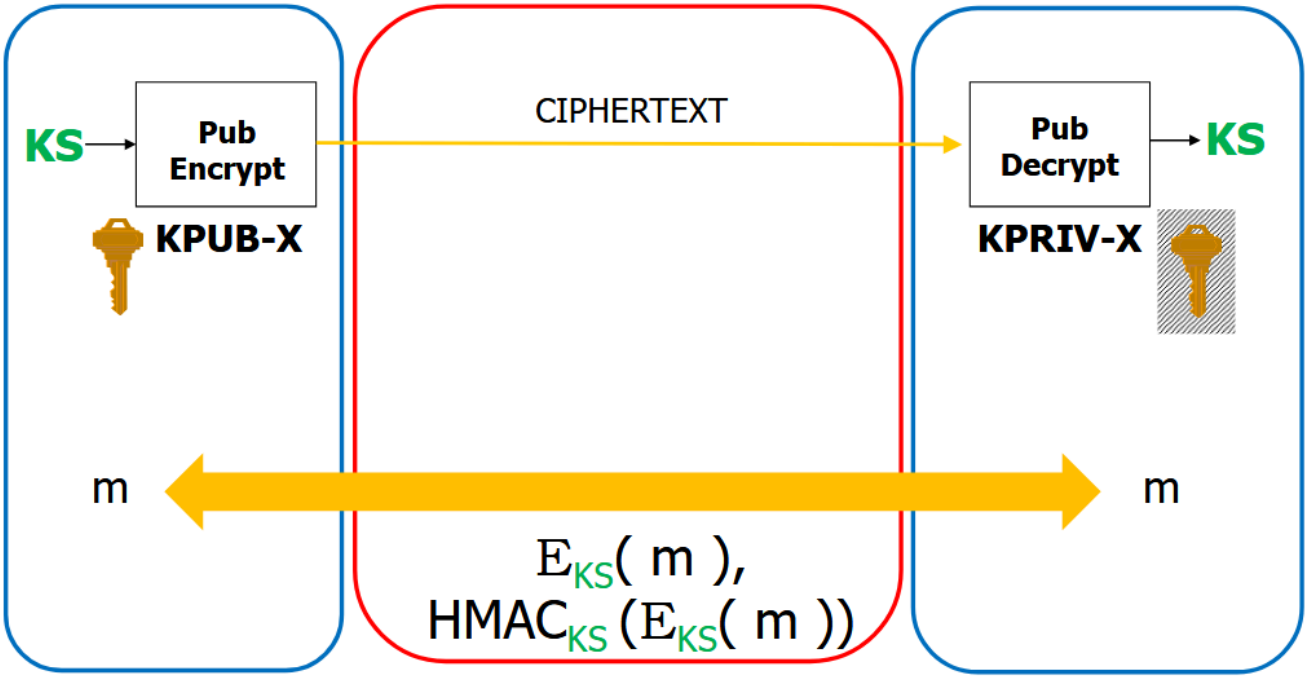
Esempio e Cenni Storici. Gli algoritmi moderni di crittografia a chiave pubblica sono stati inventati nel 1976 da Diffie e Hellman, Markle da un punto di vista *concettuali*, tuttavia a livello implementativo non erano riusciti a capire quali alogirtmi usare. Dopodiché Rivest, Shamir e Adleman sono riusciti a inventare l'algoritmo *RSA* che implementa la crittografia a chiave pubblica con grande successo e infatti sono diventati ricchi 🤑. In realtà, si è scoperto che uno studento li aveva inventati prima di loro in un tirocinio estivo in un'organizzazione di intelligence britannica, tuttavia per motivi di lavoro non poteva pubblicarlo.

Osservazione. La *"chiave privata"* della crittografia a chiave pubblica non è correlata in nessun modo alla *"chiava privata"* della crittografia a chiave privata.

Terminologia. La *"crittografia a chiave asimmetrica"* è sinonimo di *crittografia a chiave pubblica*, invece la *"crittografia a chiave simmetrica"* è sinonimo di *crittografia a chiave privata*. I termini appena detti sono utilizzate nelle normative italiane.

Osservazione. La crittografia a chiave pubblica è più lenta della crittografia a chiave pubblica in 1-2 ordini di grandezza (x10-100)

Osservazione. Non garantiscono comunque integrità, né tantomeno l'autenticazione; bisognare usare HMAC!



Certificati e Certification Authority

X

Problema motivante: distribuzione delle chiavi pubbliche. Certificati e Certification Authority: definizione di certificato, certification authority. Procedura di rilascio dei certificati da parte dei CA, due casi. Esempi: DNS-name e firme digitali.

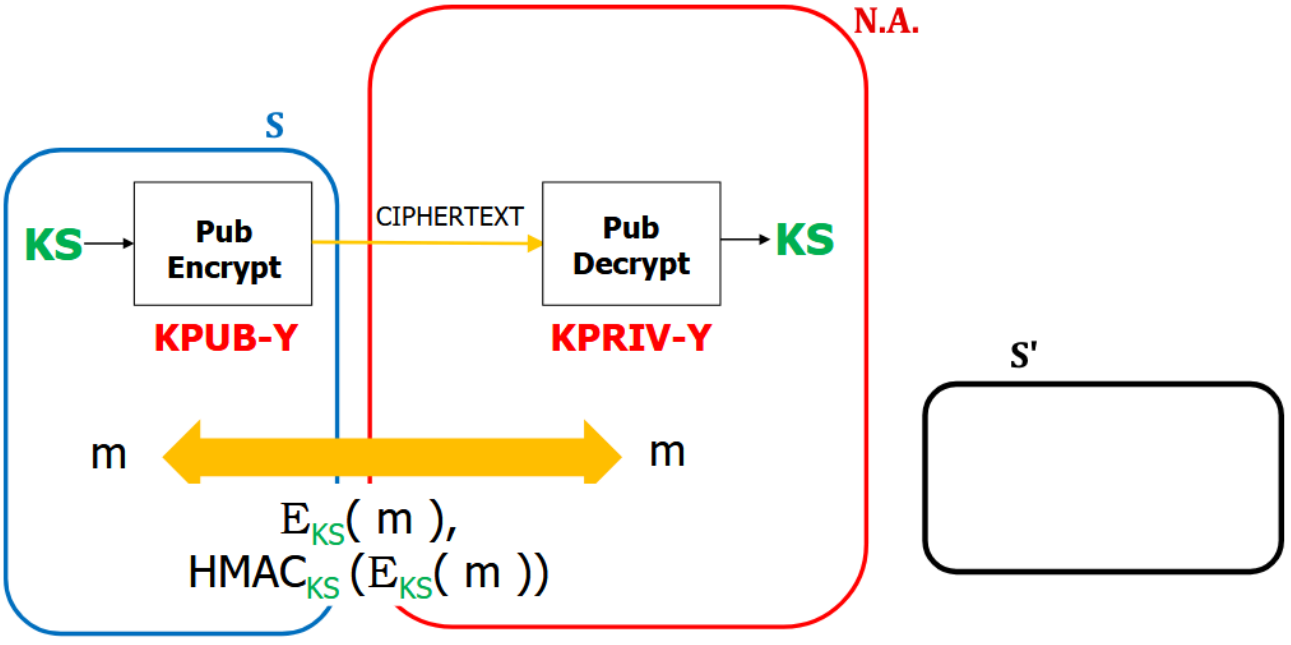
X

0. Voci correlate

- Crittografia a Chiave Pubblica
- Introduzione alla Network Security

1. Distribuzione delle Chiavi Pubbliche

Osservazione. Col strumento della crittografia è chiave pubblica è possibile realizzare le proprietà di sicurezza. Tuttavia, ciò assume che di base le chiavi pubbliche vengano *distribuite*! Questo costituisce un altro scoglio nel nostro cammino, siccome il network attacker è in grado di *distribuire delle chiavi pubbliche "false"* annullando così tutte le proprietà di sicurezza realizzate.



Una soluzione teorica al problema è quella di usare un *canale sicuro aggiuntivo*; tuttavia, questa è impossibile.

Vediamo dunque un *approccio moderno* al problema.

Approccio. Sia DNS-Name il *Subject*, e il server che *"possiede"* il DNS name deve distribuire l'associazione $(\texttt{DNS-name}, K_{\text{PUB}})$. Gli step che deve realizzare sono le seguenti:

- *Punto d'arrivo*: il Subject Fisico S' è in grado di comunicare la coppia ($\texttt{DNS-name}, K_{\text{PUB}}$) sul canale non sicuro al Subject Fisico S, che poi verificherà se e valido o meno *"per conto suo"*, e nel caso di esito positivo concordano la *chiave privata simmetrica*. In questo caso, il network attacker sarà solo in grado di impedire la comunicazione, siccome non sarà né in grado di alternare né di falsificare la coppia che *"certifichi"* il diritto di utilizzare un certo subject.
- *Punto di partenza*: Il server S' è *"configurato in modo sicuro"* a priori e ha usato un *canale sicuro di comunicazione* solo una volta a priori
- Poi vediamo di mettere tutto assieme!

L'approccio si generalizza anche su *subject* di tutti i tipi, per ora ci focalizziamo di questo tipo.

X

---

2. Certificati e Certification Authority

Vediamo adesso il *"punto di partenza"* della scaletta stilata prima, per affrontare il problema della distribuzione delle chiavi pubbliche. Facciamo un paio di definizioni preliminari.

Certificato, Certification Authority

Un *certificato* è un *file* che contiene la coppia ($\texttt{SUBJECT}, K^{S}_{\text{PUB}}$) codificato mediante dei protocolli specificy; inoltre contengono il nome dell'*entità* che garantisce la veridicità dell'associazione (quindi ad essere pignoli è una tripla).

L'entità che garantisce la veridicità sono le *certification authority*, e sono le stesse entità che generano i *certificati* (in linea di principio...)

Il Certification Authority garantisce quindi la seguente catena di legami, mediante il certificato:

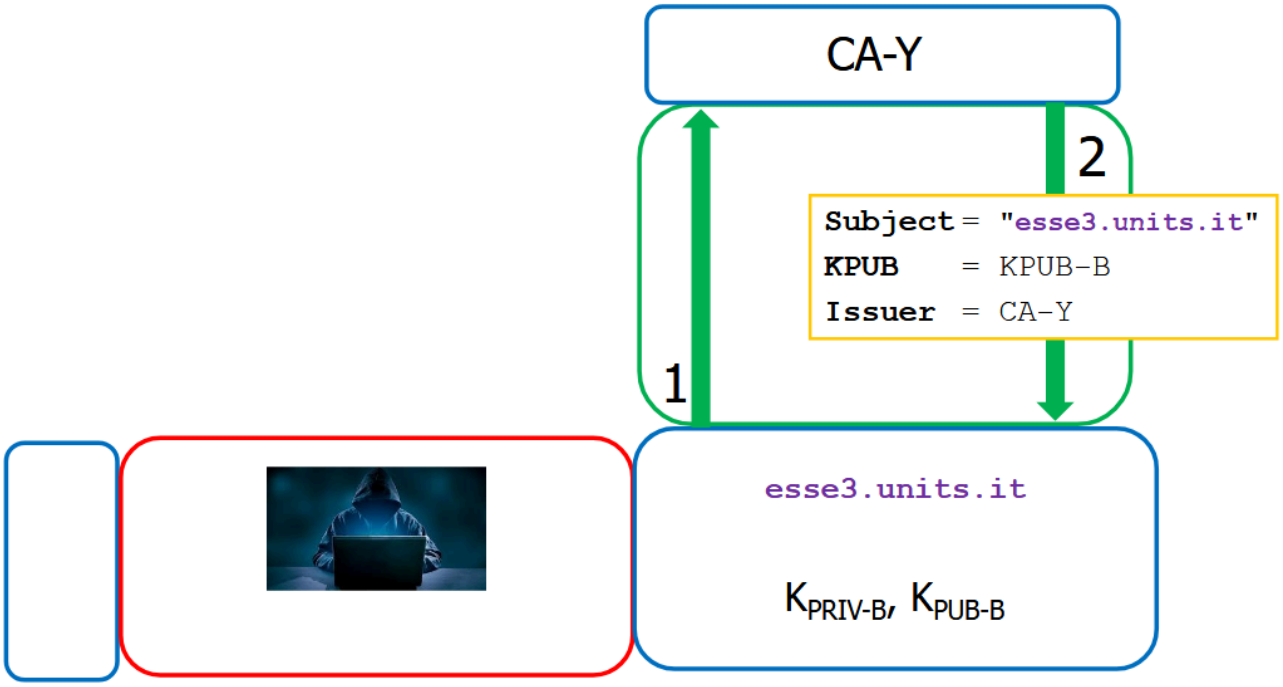
$$\texttt{Subject Fisico} \longleftrightarrow \texttt{Subject} \longleftrightarrow K^{(S)}_{\text{PUB}}$$

Ovvero:

- $\texttt{Subject Fisico} \longleftrightarrow \texttt{Subject}$ vuol dire *"Subject Fisico ha diritto di utilizzare una certa stringa come Subject*"
- $\texttt{Subject} \longleftrightarrow K^{(S)}_{\text{PUB}}$ vuol dire *"Subject deve conoscere la chiave privata in relazione a $K^{(S)}_{\text{PUB}}$.

Q. Supponendo che un server abbia la coppia chiave privata e pubblica, come può il server S (il subject fisico) a farsi rilasciare un *certificato* da una *certification authority*?

Generalmente, il *certification authority* prima si accerta che il *Subject Fisico "abbia diritto"* di autenticarsi come *Subject X*, poi nel caso positivo dell'accertamento il CA rilascia il *certificato* mediante un *canale di comunicazione sicuro*.



Notiamo che quindi in questo caso il server ha *"utilizzato una sola volta"* un canale sicuro di trasmissione.

Q. In che senso *"avere diritto"* di identificarsi come Subject X? Come si fanno i controlli? Come si accerta che il subject fisico sia chi dice di essere?

Purtroppo questi dettagli dipendono dall'*uso del certificato*, facciamo dei cenni.

Esempio. (*DNS name*)

1. CA comunica un *numero casuale* al Subject Fisico
2. Il Subject Fisico crea un *RR* di tipo *TXT* di nome *DNS-name* con valore del numero casuale comunicato prima
3. Il CA effettua una resolution in *DNS-name* con tipo *TXT*. Se lo trova, OK.

Esempio. (*Firma digitale con validità legale*)

1. Il CA incontra il subject fisico in persona o mediante una call telematica ed effettua un controllo della documentazione personale

Osservazione. Notiamo che abbiamo sempre assunto che all'inizio il subject fisico abbia la coppia *chiave privata-chiave pubblica* già all'inizio. Nella realtà abbiamo un altro caso molto più comune, ossia è il Certification Authority a rilasciare la *coppia delle chiavi* e poi li fornire al subject fisico, poi per cancellarli.

Comunque esistono i casi in cui il Subject Fisico abbia già la *key pair*, in questo caso li mette in un file di formato *"Certificate Signing Request"* e poi lo invia al CA e richiede di verificarlo.

Distribuzione e Validazione dei Certificati

X

Distribuzione e validazione dei certificati: problema, ipotesi "magica" (cenno).

X

0. Voci correlate

- Certificati e Certification Authority

1. Distribuzione e Validazione dei Certificati

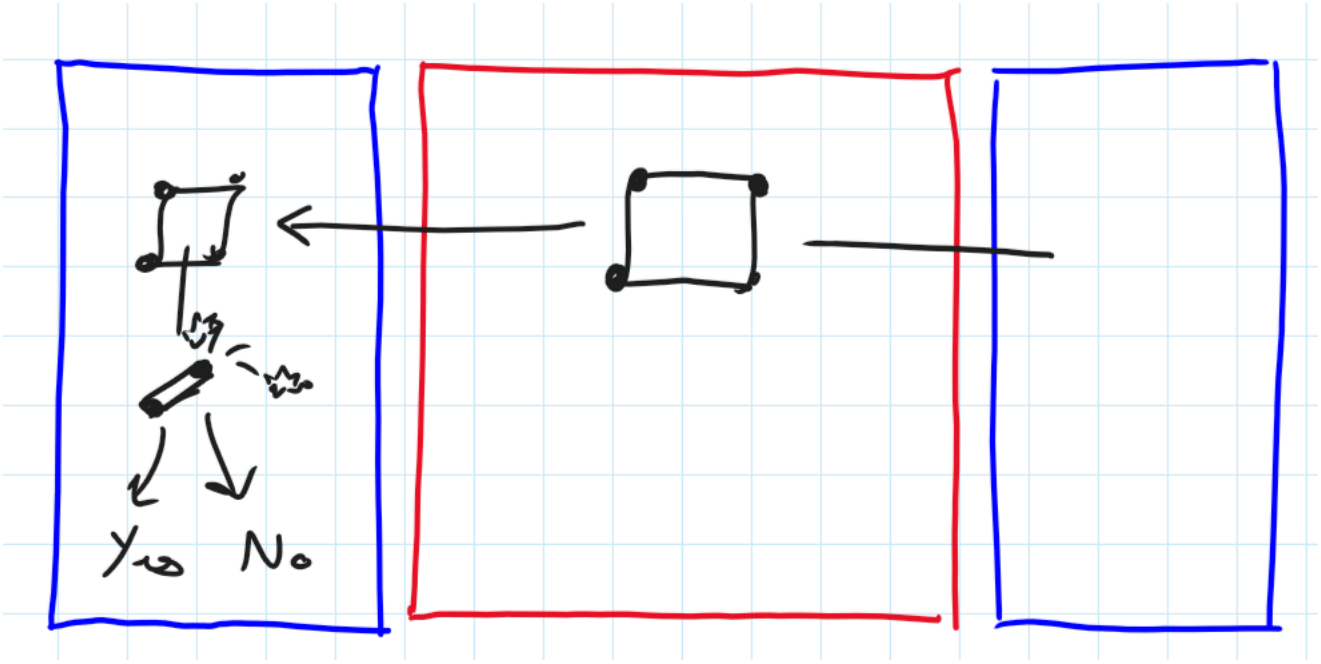
Supponiamo che una CA abbia rilasciato un *certificato* che attesti il diritto del subject fisico di riconoscersi come subject X, che a sua volta conosce la chiave privata.

Q. Come si distribuiscono i certificati?

Siccome è possibile avere ulteriori inghippi con l'attacker, sarebbe da stare attenti siccome si potrebbe falsificare o modificare il certificato, annullando dunque l'*autenticità* e l'*integrità*.

Per risolvere il problema, facciamo la seguente ipotesi *"magica"*

> **Ipotesi Magica**
>
> Si suppone che, dato un *certificato*, ogni subject è in grado di verificare che sia *autentico* e *integro* con uno *"strumento magico"*, e può farlo indipendentemenata dal canale dal quale *"esce"* il certificato.

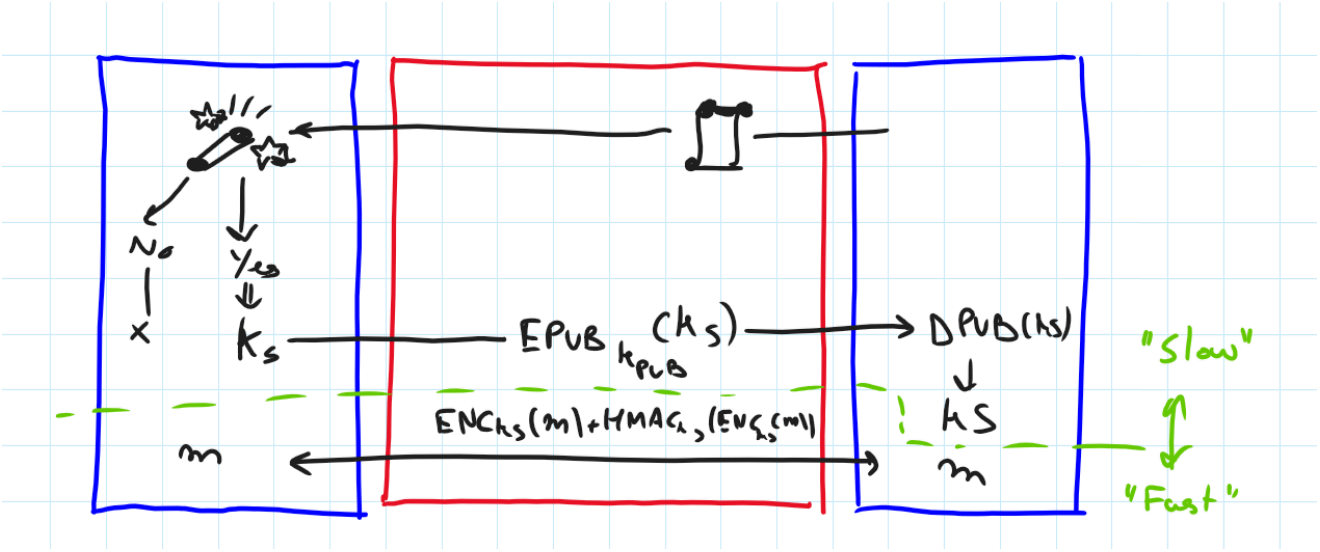


Vedremo dopo come sarà possibile implementare questo *"test magico"*.

Osservazione. Notiamo che supponendo pure vera questa ipotesi, è comunque possibile che le CA rilascino dei *"certificati fasulli"* (per errore o frode); in questo caso, facciamo finta di nulla e assumiamo che *"Ogni CA dice sempre la verità"*. Per quanto riguarda la praticità, vediamo dopo....

Osservazione. Alla fine, siamo riusciti a realizzare un percorso che implementi le *funzionalità di sicurezza* sui flussi di byte. Ad esempio, un procedimento è come segue:

1. Server ottiene un *certificato* con una chiave pubblica, poi inoltre possiede la chiave privata associata
2. Server manda a Client il *certificato*
3. Client verifica il certificato
4. Nel caso positivo, client sceglie un *valore* per la *chiave simmetrica* e lo comunica al Server mediante crittografia a chiave pubblica
5. D'ora in poi Server e Client si comunicano mediante la *crittografia a chiave privata*



Notiamo che le parti 1-3 è rappresentano parte *"lenta"* dela comunicazione, e viene usata solo per scegliere la chiave simmetrica; invece le parti 4-5 sono *"veloci"* e si usano per comunicare i *dati*.

Il procedimento appena descritto è una *tecnica generale* di cui si basano *"quasi"* tutte le applicazioni pratiche degli strumenti matematici appena descritti. Infatti:

- Rendiamo il canale non sicuro in sicuro con la *crittografia chiave privata*
- La comunicazione per concondare la *chiave privata simmetrica* viene fatta con la *crittografia a chiave pubblica*
- Si distribuiscono le chiavi pubbliche su *canali non sicuri* in *"modo sicuro"* mediante i *certificati*

Osservazione. I certificati sono pubblici... per forza; non c'è nulla di segreto

Esempi. Vediamo alcuni *standard* di certificati:

- X509: Molto comune e hanno validità legale in Italia
- PGP: Meno comune, presenti solo su alcuni software

Osservazione. La verifica dei certificati è ancora più complessa, infatti ci sono almeno altri due aspetti da vedere (non li vedremo, adesso accenniamo):

- *Data di scadenza*: ogni certificato ha una *periodo di validità* da una data X a data Y; se il software nota che il *"giorno corrente"* non sta nel periodo di validità, allora ritiene invalido il certificato
- *Scopo*: ogni certificato è rilasciato per *alcuni usi*, ad esempio un certificato per *"server auth"* si ritiene invalido per uso di *"digital signature"*.

Crittografia a Chiave Pubblica in Pratica

X

Aspetti pratici della crittografia a chiave pubblica. Concetto intuitivo di Trust Set e Key Set, test di validità dei certificati. Esempio fondamentale: certificato valido e certificato non valido. Significato di certificato non valido in pratica. Esempio di certificati non validi: certificati HTTPs scaduti. Definizione formale di Trust Set, Key Set e Personal Set.

X

0. Voci correlate

- Certificati e Certification Authority
- Crittografia a Chiave Pubblica
- Introduzione alla Network Security

1. Aspetti Pratici della Crittografia a Chiave Pubblica

Premessa. Ogni certificato in arrivo proviene da un *canale non sicuro*, non lo disegneremo esplicitamente.

In pratica, non è vero che *"ci fidiamo"* sempre delle Certification Authority o che sia *sempre possibile* effettuare il test di validità di un certificato. Nella realtà:

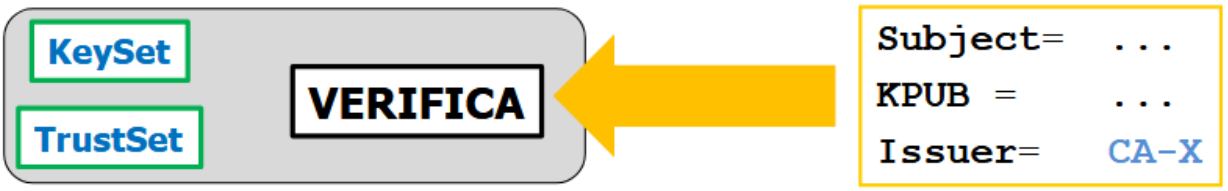
- Ci fidiamo solo dei *certificati* prodotti da *alcuni Certification Authority*, in particolare devono trovarsi all'interno dell'insieme *"Trust Set"* (vediamo dopo cos'è veramente)
- Possiamo eseguire il test di accertamento delle certificazioni solo quelli prodotti da *alcuni Certification Authority*, in particolare devono stare nel *"Key Set"*

Gli insiemi sono delle *strutture dati* predefinite: infatti, ricordiamo che i subject sono *"configurati in modo sicuro"*, e quindi vuol dire che gli insiemi Trust Set e Key Set siano *"correttamente"* definiti.

Un software esegue il seguente algoritmo per verificare un certificato:

Algoritmo.

1. CA-x sta nell'intersezione tra Trust Set e Key Set?
2. Se sì, proseguire; altrimenti abortire il programma
3. Effettuare il test sul certificato
4. Se sì, finire e usare il certificato



Osservazione. Notiamo che CA-x non sta in Trust set o Key Set vuol dire che *potrebbe* essere falso o che *non posso effettuarci il test*, ma non vuol dire necessariamente che sia veramente falso o che sia veramente modificato o che non sia autentico.

Osservazione / Richiamo. Se un certificato è *valido* allora si garantisce la seguente catena di associazioni:

$$\texttt{Subject Fisico} \longleftrightarrow \texttt{Subject} \longleftrightarrow K_{\text{PUB}}^{(S)}$$

Quindi, ragionando in maniera completa:

- Chi ha richiesto il certificato ha il diritto di usare Subject come identificatore
- La chiave pubblica $K_{\text{PUB}}^{(S)}$ è veramente la chiave pubblica del Subject

Q. Se invece un certificato *non* è valido? Cosa vuol dire?

Vuol dire che uno dei test descritti nell'algoritmo (^8dc091) ha fallito:

1. CA non sta nel Trust Set
2. CA non sta nel Key Set
3. CA sta nel Key Set ma il test ha restituito un esito negativo

Ovvero non ho più la seguente catena di garanzie:

$$\texttt{Subject Fisico} \not\longleftrightarrow \texttt{Subject} \not\longleftrightarrow K_{\text{PUB}}^{(S)}$$

In pratica consideriamo la possibilità che il *Subject Fisico* non abbia veramente diritto di identificarsi come *Subject X*.

Q. Cosa fa il software quando riceve un certificato non valido?
Dipende dalla configurazione del software, fa uno tra le due opzioni:

1. Abortisce l'esecuzione del programma (procedimento *"matematico"*)
2. Chiede all'utente se procedere, usando il certificato potenzialmente falso (*caso più comune*)

Esempio. Un caso piuttosto comune è quello di vedere dei *certificati scaduti* su HTTPs, e l'utente può scegliere di connettersi col web server nonostante l'avviso del certificato invalido.

Osservazione. La *"non validità* o *"validità"* di una certificazione è una proprietà relativa alla configurazione del nodo di subject, siccome è possibile definire insiemi Trust Set o Key Set diversi.

2. TrustSet, KeySet e PersonalSet

Vediamo di definire formalmente il concetto di *TrustSet* e *KeySet*. Vedremo inoltre la necessità di definire l'insieme delle *"proprie chiavi"*, chiamato *PersonalSet*.

TrustSet e KeySet

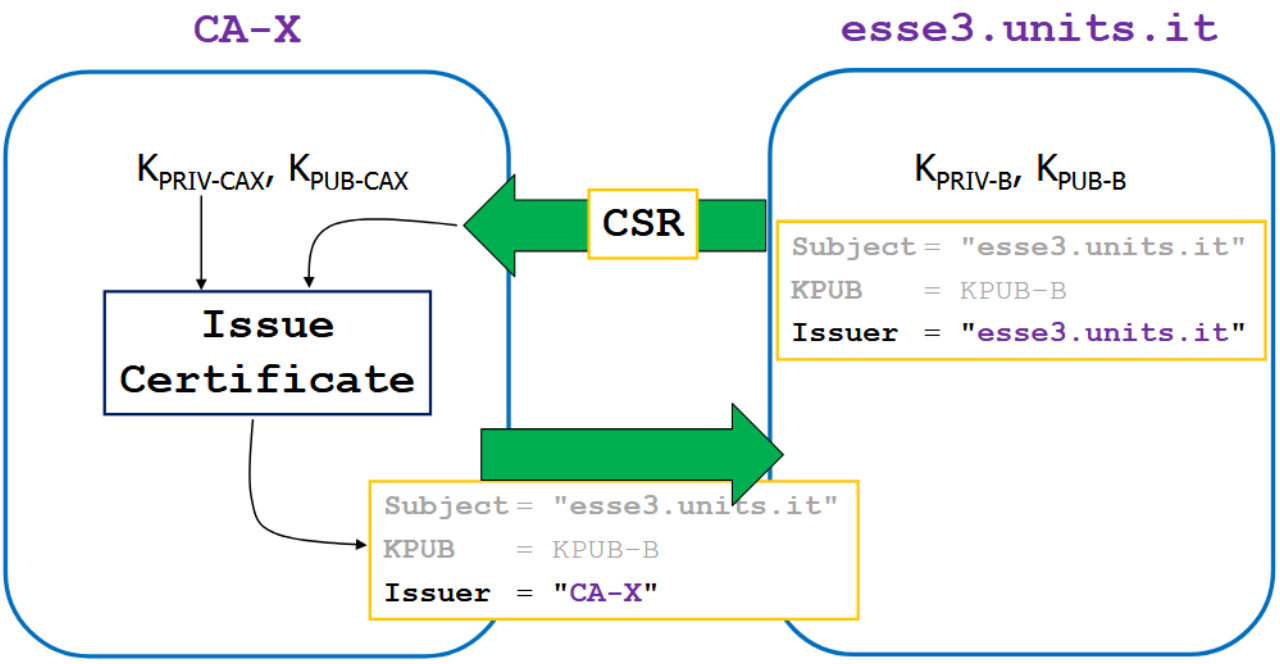
TrustSet e KeySet sono *insiemi di certificati autofirmati* con chiave pubblica.

Capiremo perché siano *"self-signed"* dopo...

Gli insiemi TrustSet e KeySet stanno in una cartella che dipende dal *sistema operativo* o dal *software specifico*. Ad esempio, in openssl si ha un file con una sequenza di certificati autofirmati.

Adesso vediamo cosa vuol dire *"configurazione sicura"* dei subject nuovamente, tornando al caso *"poco comune"* di rilascio certificati:

- Il server genera un *certificato autofirmato* e lo invia al CA
- Il CA effettua dei controlli, e nel caso di esito positivo genera un *certificato* con una sua *chiave privata* (capiremo meglio perché)



Una volta che il certificato viene mandato da parte del Certification Authority, il subject server la preserva assieme alle chiavi associate in una struttura dati chiamata *Personal Set*.

Personal Set

Un *personal set* è un insieme delle *"proprie chiavi"*, ovvero ogni elemento consiste nella

> tripla ($K_{\texttt{PUB}}$, $K_{\texttt{PRIV}}$, `CERTIFICATE`).

I *PersonalSet* sono salvati come dei *"file sparsi"*.

Protocollo TLS

X

Applicazione dei strumenti matematici per realizzare un canale di comunicazione sicuro: protocollo TLS. TLS da un punto di vista funzionale: definizione e proprietà. Osservazioni: il lato server dimostra di conoscere la sua chiave privata ma non lo comunica, prima istanza di autenticazione lato server. Esempio di protocollo TLS applicato: HTTPs, aspetti funzionali. Implementazione di TLS: procedimenti, osservazioni.

X

0. Voci correlate

- Introduzione alla Network Security
- Crittografia a Chiave Pubblica
- Crittografia a Chiave Privata e HMAC
- Crittografia a Chiave Pubblica in Pratica
- Certificati e Certification Authority
- Comunicazione tra Processi

1. TLS

Applichiamo tutti gli strumenti matematici appena visti per realizzare un canale di comunicazione sicuro.

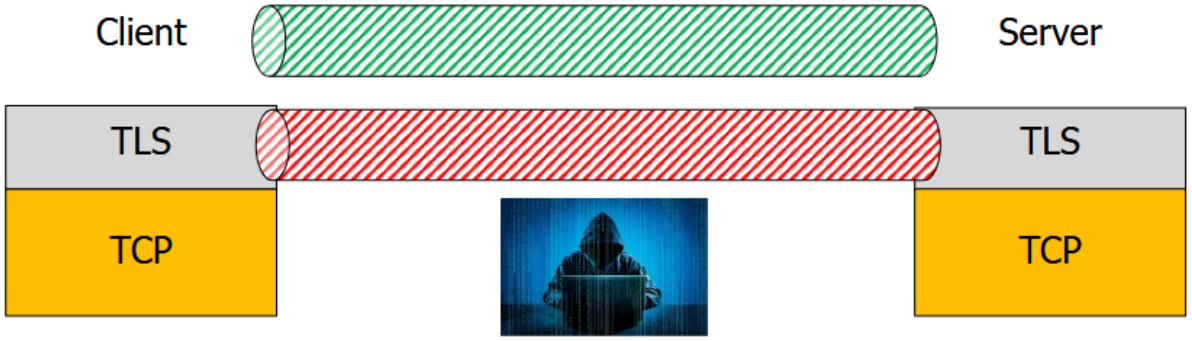
1.1. Aspetti Funzionali

Lo vediamo prima di tutto da un punto di vista *"funzionale"*, ovvero *"come si usa"* TCP.

> **TLS (Transport Layer Security)**
>
> *TLS* è un protocollo che usa delle techniche *"crittografiche"* che *"irrobustire"* le connessioni TCP, che a loro volta sono dei *canali di comunicazione vulnerabili*.

Cenni Storici. Inizialmente si chiamava SSL, dopodiché c'erano varie versioni fino ad oggi con TLS.

Intuitivamente, TLS è un *"layer aggiuntivo"* alla pila dei protocolli Internet.



A livello di implementazione, *"basta"* riscrivere i programmi (le librerie) in un modo da poter essere compatibili con TLS. Nei dettagli più specifici, bisogna consultare gli RFC.

Proprietà. Il protocollo TLS è in grado di garantire le seguenti proprietà di sicurezza:

- *Secrecy*: Si usa la *chiave privata simmetrica* per comunicare dati e la si cambia di frequente
- *Integrity*: Se un lato della connessione si *"accorge"* che i dati vengono alterati in transito, si chiude immeditamente la sessione.
- *Authentication (Server)*: Si usa un *certificato* sia per verificare se il *server che comunica abbia veramente diritto di usare il DNS name* e con la chiave privata si dimostra che nel *lato server ci sia veramente il lato server*.

Osserviamo che non si garantisce la *availability*, anzi *"viene di meno"*. Infatti, un possibile attacco è quello di alterare leggermente i bit nella comunicazione in una maniera *"costante"*, impedendo la comunicazione tra i due subject (attacco noto come *"Denial of Service"*).

Osservazione. Se si usano certificati invalidi, crolla *solo* la certezza del DNS-name.

Osservazione. L'attaccante non può impersonare il lato server, infatti deve dimostrare di conoscere la *chiave privata giusta*. Inoltre, notiamo che il *server* dimostra di conoscere la chiave privata, ma non lo comunica! Questo è una differenza dal caso di *client-side authentication*.

Osservazione. Questo è il primo esempio in cui si autentica il *server*.

Esempio. (*HTTPs*)
HTTPs è un primo esempio di protocollo applicativo che comunica su TLS. Da un punto di vista funzionale, il browser:

1. Risolve DNS-name
2. Si connette al server con porta 445
3. Usa funzioni send()/receive() con proprietà TLS

1.2. Aspetti Implementativi

⚠ **Argomento Semplificato**

> **N.B.** Questo argomento è una semplificazione della realtà, adeguata solo per scopi didattici.

Per vedere l'implementazione di TLS vediamo *quali siano le funzioni* in TLS.

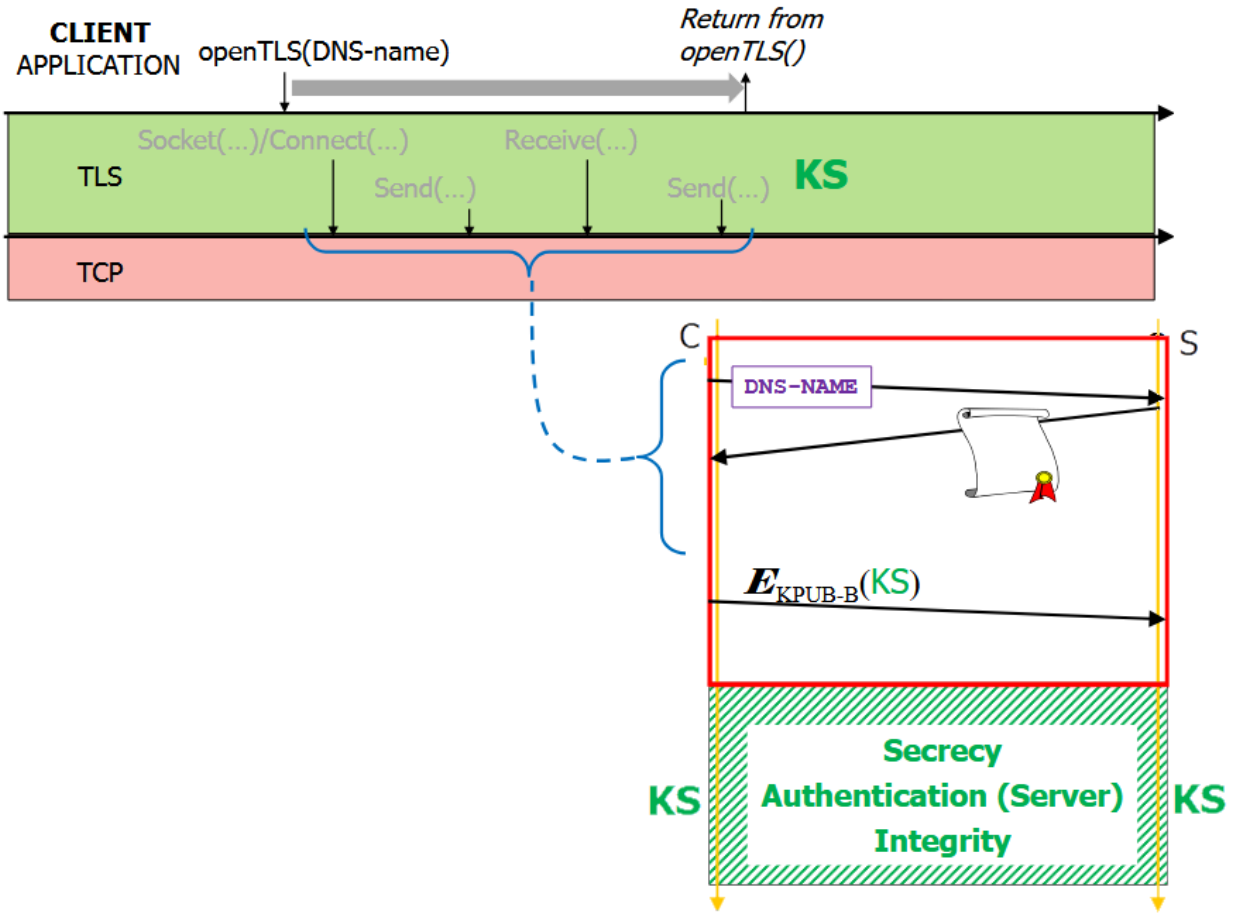
openTLS(DNS-name): Prima di tutto, per aprire una *connessione TLS* si effettuano gli seguenti step:

- Il Client invia il *DNS-name* dato in configurazione
- Server invia il *certificato* associato a *DNS-name*
- Client verifica il certificato e *anche il subject* del certificato (!)
- Client sceglie e cripta la chiave simmetrica K_S, lo invia al server

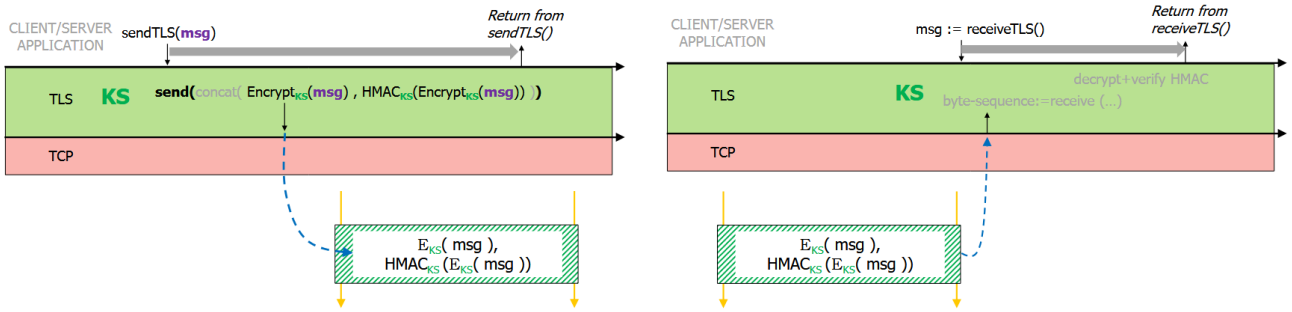
Q. Come mai il lato client deve ri-inviare il DNS-name?

Questo perché il server potrebbe gestire più domini, quindi conoscendo il DNS-name sa quale certificato inviare al lato client. Esempio: web hosting.

Altrimenti, se il client accettasse un qualsiasi certificato valido allora il network attacker *"man in the middle"* potrebbe inviare un certificato valido ma che *non centri nulla* col DNS-name richiesto.



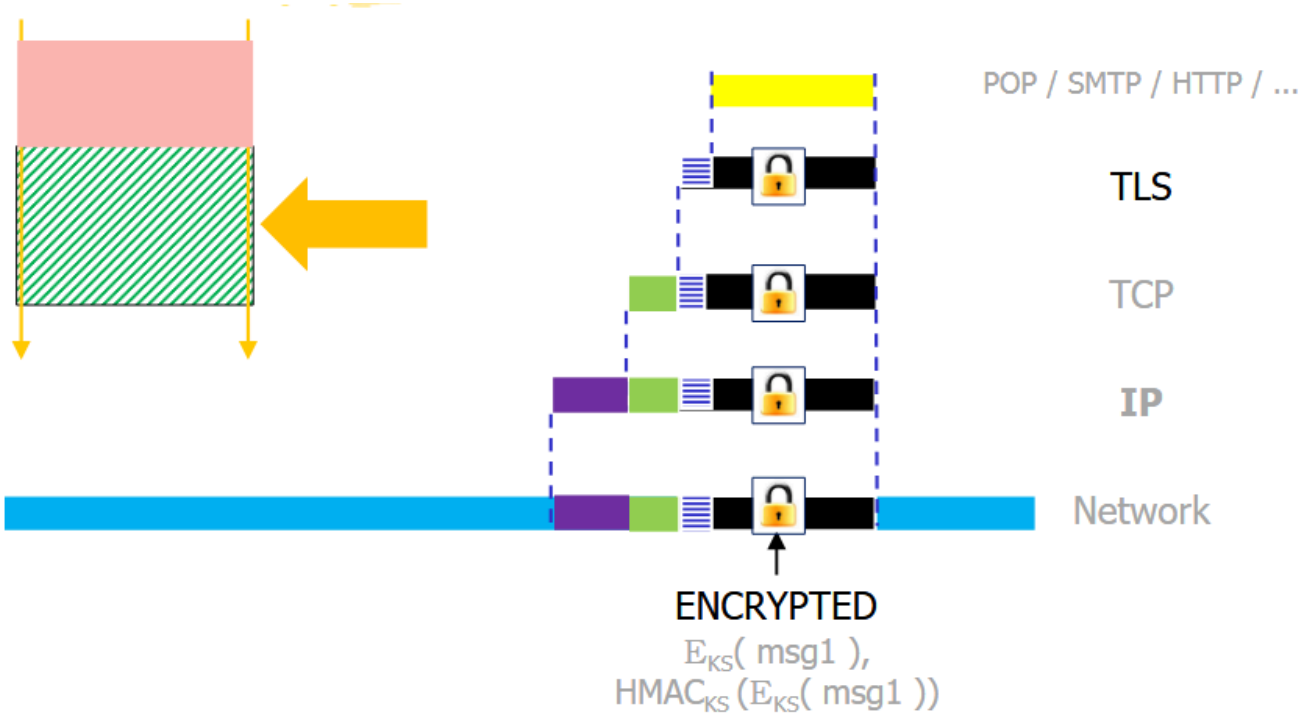
send(m) / receive(m): Una volta eseguito `openTLS(DOMAIN-NAME)`, si può iniziare a trasmettere dati. In particolare, con `send(m)` invia il messaggio criptato con chiave simmetrica e con HMAC, invece con `receive(m)` riceve il messaggio, decritta il messaggio e verifica i valori HMAC.



Vediamo meglio un paio di aspetti di TLS, essendo consapevoli dell'implementazione:

Osservazione. Il web server conosce di dimostrare il suo *"segreto"* conoscendo la sua chiave simmetrica.

Osservazione. Si ha un ulteriore strato di incapsulamento del pacchetto applicativo, fatto durante la *fase di apertura* della connessione TLS.



× Bocciatura Automatica

Soltanto il *payload del protocollo applicativo* è criptato, non tutto il payload del frame network! Se lo fosse crittato tutto, i router non sarebbero grado di capire *come* instradare il pacchetto...

Osservazione. Come funzionerebbe TLS con *payload proxy*? In TLS assumiamo che c'è solo una connessione TLS... mistero!

Osservazione. Una *"buona pratica"* da assumere come proprietà è quello di cambiare la chiave simmetrica ad ogni connessione TLS su TCP, ovvero in ogni connessione TLS dev'essere invocata la funzione `openTLS()` nuovamente, indifferente dal fatto che prima si abbia già aperto una connessione TLS e quindi avere già una chiave simmetrica concordata.

TLS in Pratica

X

Aspetti pratici di TLS. Difesa contro DNS spoofing. Uso di HTTPs: livello di riservatezza, rubare cookie a siti HTTPs aggirando le difese TLS (crittografia).

X

0. Voci correlate

- Protocollo TLS
- Aspetti Strategici del DNS
- Protocollo HTTP
- Sessioni HTTP
- Proprietà di Sicurezza

1. Attacchi Significativi TLS

Conviciamoci che il *protocollo TLS* funzioni veramente per difendersi dagli network attacker, garantendo le proprietà di sicurezza di *riservatezza, integrità e autenticazione (lato server)*. Lo facciamo presentando un esempio ed esplorando tutti gli esiti possibili

DNS Spoofing. Supponiamo che un *processo client* si colleghi ad un *processo server* via TCP su TLS. Supponiamo che l'RR per cui si identifica il nodo del processo server sia `www.a.com A IP-A`

Adesso supponiamo che il *network attacker* sia in grado di *"falsificare"* la risposta DNS, portando dunque al *processo client* di contattare in realtà `IP-other`.

Cosa può fare, per *"comunicare"* col client?

1. Inviare un certificato con DNS name diverso da `www.a.com`
2. Possedere un *"certificato valido"* per `www.a.com` ma con chiave pubblica del network attacker
3. Possedere un certificato *"proprio"* con issuer *CA-s* che appartiene al *trust set* del client
4. Come 3. ma *CA-s* non sta nel *trust set* del client

Notiamo che ogni tentativo fallisce, infatti:

1. Il client si accorge la discrepanza tra i subject DNS-name, termina quindi la connessione
2. Impossibile, si assume che i certificati prodotti da una CA che sta nel *trust set* siano sempre *"veritieri"*
3. Impossibile, come sopra
4. Fallisce in quanto il client rileva l'invalidità del certificato

Vediamo che quindi con TLS il *browser* è in grado di accertare che stia veramente comunicando col *"proprietario legittimo"* del name server dell'URL contenuto nell'address bar!

Osservazione. Notiamo che comunque rimane una problematica irrisolta legata al *sistema DNS*, ovvero il network attacker potrebbe acquistare un *dominio DNS* con nome *"simile"* a `www.a.com` (come `www.a_.com`) e comprarci dei certificati validi (che è possibile, siccome in questo caso è vero che NA possieda il dominio DNS *"ingannevole"*) e quindi infine riuscire ad ingannare il client di comunicare con `www.a.com`.

Tuttavia, tutte le proprietà di sicurezza funzionano via HTTPs.

X

2. Uso di HTTPs

Vediamo adesso le *implicazioni pratiche* di applicare lo "strato" *TLS* su comunicazione *HTTP*.

Come in HTTP, ogni documento è identificato da un URL, ed il protocol name (la prima parte) diventa `https`. La porta su cui si effettua la comunicazione HTTPs è 443. Fino ad oggi, HTTPs ha *"quasi sostituito"* il protocollo HTTP; si stima infatti che in Europa l'uso dell'HTTPs è previsto nel 95.7% dei casi [1].

Abbiamo visto che HTTPs garantisce le proprietà di sicurezza previste da TLS, ovvero *riservatezza, autenticazione ed integrità*. In altre parole, HTTPs utilizza *tecniche crittografiche* per *"irrobustire"* una connessione TCP, che è invece *vulnerabile*.

2.1. Proprietà di Sicurezza di HTTPs

Vediamo di approfondire bene le proprietà di sicurezza.

Q. Se visito un *server* con un certo nome su *TLS/HTTPs*, è possibile che il Network Attacker sappia della visita?

La risposta è sì, per almeno due motivi:

- Nella fase di apertura della comunicazione *TLS* (ovvero il client invoca la funzione `open(TLS)`) il client deve inviare il *DNS-name* al server
- Il Network Attacker può tracciare le comunicazioni *DNS*, quindi vedere quali nome vuole risolvere il client

Q. Invece per pagine specifiche? E' possibile che il Network Attacker conosca la *local URL* specifica?

La risposta è *no* in questo caso, siccome questa informazione è contenuta solamente nelle richiesta/risposta HTTPs, che sono criptate.

Q. Come mai si deve usare HTTPs anche quando *non sono autenticato*? In questo caso, sembrerebbe fattibile l'approccio di usare HTTP.

In realtà ricordiamo che HTTPs garantisce and l'*integrità* e *autenticazione (lato server)*, quindi il *network attacker* non è in grado di *"portarmi vuole dove vuole lui"*. Oppure, è anche importante per *inviare i dati* (quindi prima di prelevare un form)! In questo modo, si evita che il *network attacker* manipoli il FORM facendoci inviare le credenziali a lui, oppure che aggiunga un *keylogger JS* per tracciare i nostri dati.

Quindi vediamo che chiaramente è importante che tutto il traffico sia in *HTTPs*, non solo per prelevare pagine di URL protetti.

2.2. HTTPs e Cookie

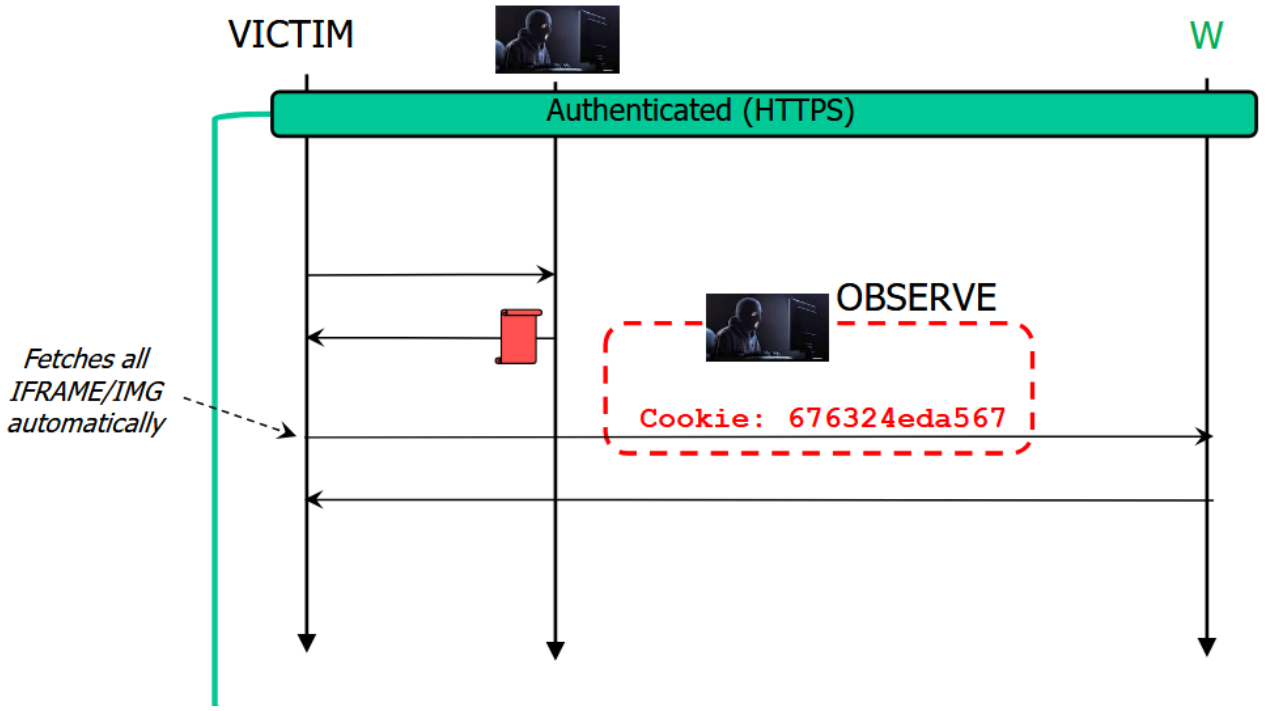
Fino ad'ora abbiamo rivisitato le proprietà di sicurezza garantite da HTTPs, evidenziando la loro importanza. Adesso vediamo un altro aspetto: i *Cookies*.

Supponiamo di avere il seguente scenario:

- Vittima è autenticata sulla webapp W
- W mantiene *sessioni autenticate* su HTTPs

Se il *network attacker* è in grado solo di osservare il traffico, non può far nulla per *"rubare"* il cookie della vittima. Tuttavia, se possedesse un *URL* controllato da lui allora potrebbe essere in grado di rubare il cookie!

La pagina web *"infettata"* del network attacker può contenere un tag *IFRAME*, *IMG* o simile in riferimento ad un URL di W, specificando il *protocollo HTTP*. In questo caso, se la sessione HTTPs è ancora mantenuta allora la vittima visiterà W *"a causa dei tag sul sito web infettato"* ed il gioco è fatto, siccome il network attacker può osservare il traffico HTTP col cookie.



Notiamo che questo è un *problema intrinseco ad HTTP*, ed il network attacker ha aggirato l'ulteriore layer di sicurezza sfruttando problema.

Una possibile difesa su questo attacco è quello di attribuire un ulteriore attributo sui cookie chiamato *"secure"*, per cui se è vera allora il *client browser* si impegnerà a non inviarlo mai via HTTP. In altre parole, bisogna estendere ulteriormente il protocollo HTTP.

Nel caso appena visto, se la vittima visità W su HTTP invia effettivamente delle richieste HTTP, però senza il cookie con attributo *"secure"*.

Firma Digitale

X

Firma digitale: ulteriore strumento matematico. Contesto, idea algoritmica della firma digitale. Osservazione: nessuna ipotesi sul canale di comunicazione. Firma digitale in pratica: autenticazione dei subject.

X

0. Voci correlate

- Proprietà di Sicurezza
- Introduzione alla Network Security

1. Contesto della Firma Digitale

Vediamo un altro strumento che applicheremo nell'ambito della *network security*, per implementare le propreità di sicurezza. Fino ad ora, abbiamo solo assicurato l'*integrità* e *autenticazione* nel contesto *TCP*, ossia le proprietà sono verificabili *solo* dalle estremità delle connessioni e *solo* durante la connessione.

Consideriamo uno scenario diverso; si vorrebbe che l'*integrità e l'autenticazione* di un flusso di byte sia *"permanente"* e *"verificabile da chiunque ed ovunque"*, generalizzando sul canale di comunicazione.

Esempio. *File* (referto medico, verbale d'esame, programma esebuibile), oppure una *mail*. Si potrebbe decidere i prelevare questi file in più modi, come:

- Memory pen
- Server con HTTPs
- altri mezzi

L'unico modo in cui possiamo usare TLC è la seconda casistica, ovvero server con *HTTPs*. Al massimo, riuscirò a garantire le proprietà di sicurezza rispetto al *browser*.

Per dare una descrizione più generale, abbiamo:

1. *Subject A* crea una sequenza di byte B, non indirizzata a nessuno in specifico
2. B viene spostato nello *spazio* e nel *tempo*
3. Dopo una quantità arbitraria di tempo chiunque può verificare l'*autenticità* e l'*integrità* di B, indipendentemente dal suo percorso di spazio o tempo.

X

2. Firma Digitale: Idea

Lo strumento matematico per lo scenario appena descritto è la *firma digitale*.

IDEA. Si ha un *algoritmo a due parti*, ove:

1. Chi apporta firma ha *chiave privata e pubblica*
2. Chi verifica la firma ha la *chiave pubblica*

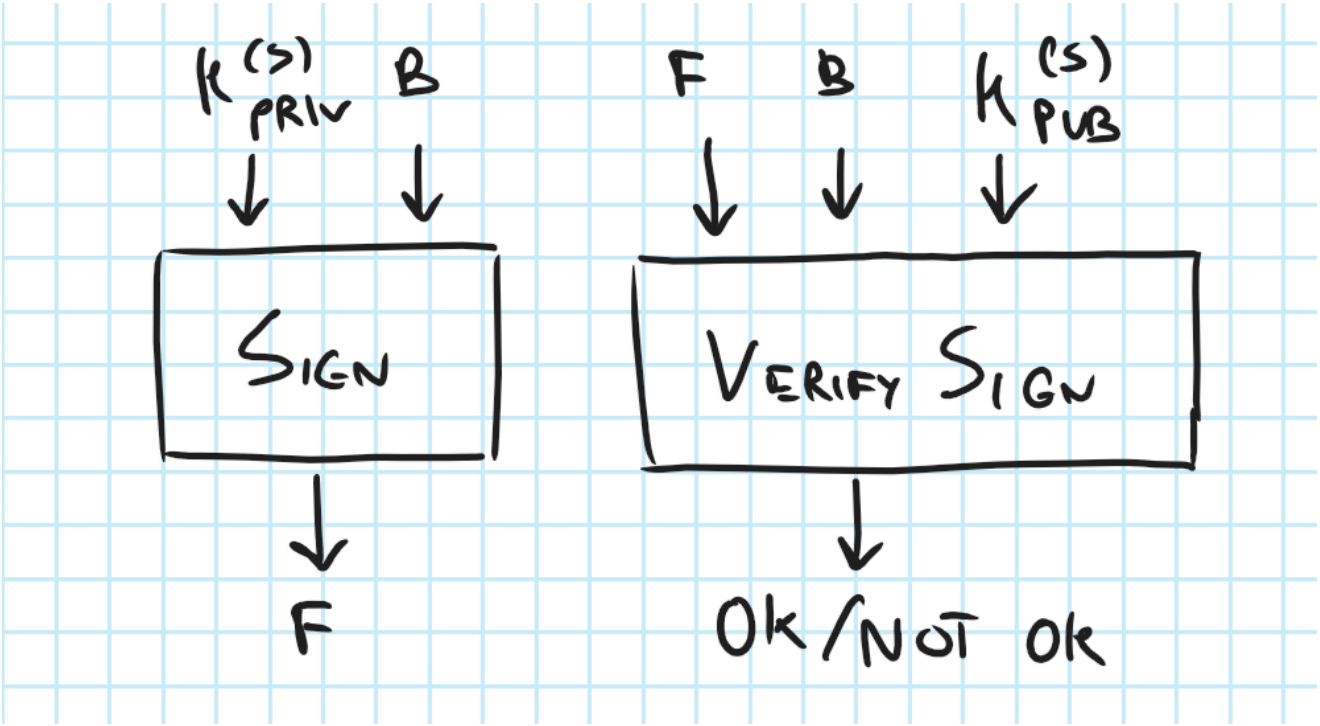
L'algoritmo di *firma* accetta B e viene parametrizzata secondo la chiava privata $K^{(x)}_{\texttt{PRIV}}$, ed in output ritorna una sequenza di byte a *lunghezza costante* (tipicamente 128 byte). D'altronde, l'algoritmo di *verifica* delle firme accetta in input *il flusso di byte e la firma* ed è perametrizzata secondo la chiave pubblica $K^{(x)}_{\texttt{PUB}}$, ed in output esce un bit (valore booleano) per indicare se la verifica è andata a buon fine o meno.

Usiamo le seguenti notazioni

- $F := \texttt{SIGN}_{K^x_{\texttt{PRIV}}}(B) \mapsto 2^N$
- $\texttt{VERIFYSIGN}_{K^x_{\texttt{PUB}}}(B, F) \mapsto \{0, 1\}$

In particolare se la verifica:

- Ritorna OK, allora la firma è *autentica ed integra*
- Ritorna NOT-OK, allora la firma o non è autentica o non è integra

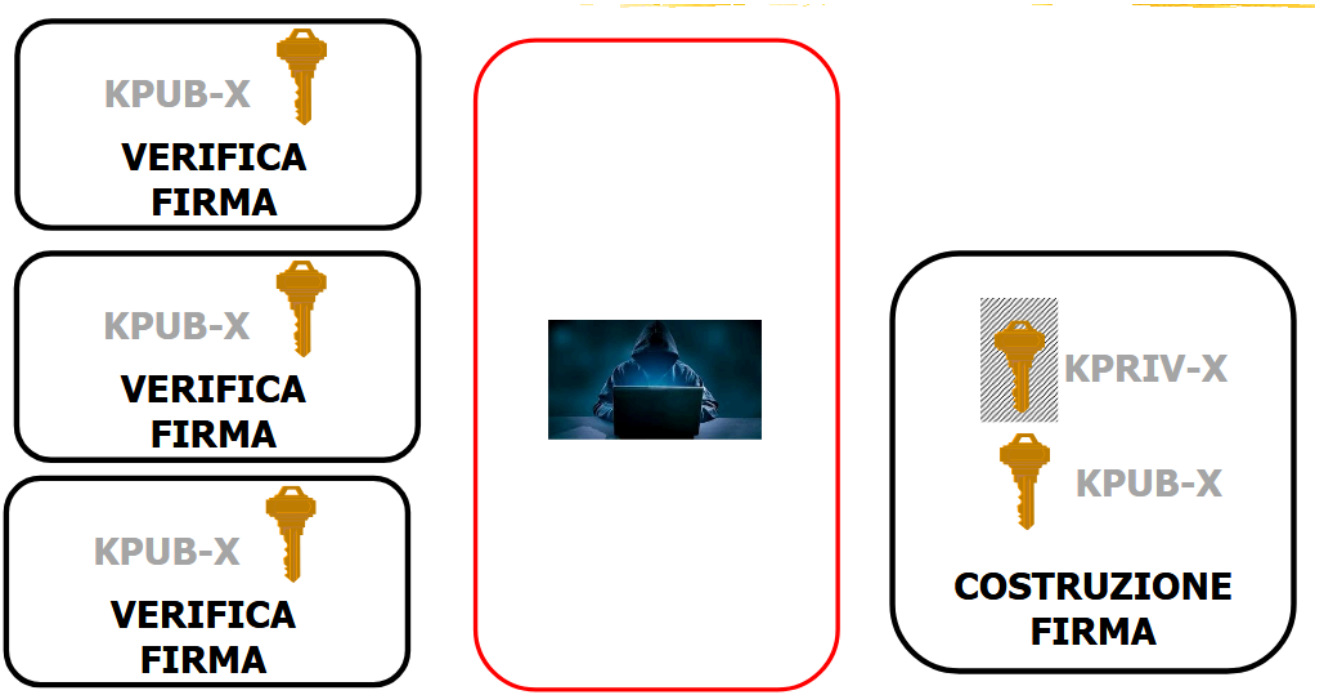


Per garantire la proprietà di integrità ed autenticazione, si dimostra che:

- Calcolare la *firma "corretta"* senza la chiave privata è *"praticamente impossibile"*
- Dedurre la *chiave privata* dalla *firma* o dalla *sequenza di bytes* è *"praticamente impossibile"*

Osservazione. Non vedremo cosa c'è dentro l'algoritmo nello specifico, vediamo solo *"come si usa"* la firma digitale. A livello implementativo, accenniamo che d'entro c'è la crittografia a chiave pubblica ed *"altro"*.

Osservazione. L'applicazione delle firme digitali richiede uno scenario di partenza in cui il subject *"verificante"* possiede la chiave pubblica del subject *"firmatario"*.



Osservazione. Chiunque può verificare, invece solo chi conosce la chiave privata può costruire la firma digitale. Questo ovviamente per garantire la proprietà di *autenticazione*

Osservazione. Fino ad ora, la *firma digitale* non richiede nessuna ipotesi sul canale di comunicazione! Al massimo, richiede la distribuzione delle chiavi pubbliche (vedremo dopo com'è realizzata). Quindi funzione anche su HTTPs, SMTP, flussi di byte *"consegnati a mano"* ed eccetera...

Notiamo che quindi il *flusso di byte* e la *firma* possono arrivare in ordine diverso, garantendo le proprietà di sicurezza lo stesso!

Aspetti Pratici della Firma Digitale

X

Aspetti pratici della firma digitale. Ulteriore requisito di proprietà di sicurezza: autenticazione sul subject fisico. Modalità di utilizzo pratiche delle firme digitali. Applicazione delle firme digitali. Considerazioni pratiche sulle proprietà di sicurezza garantite. Esempi di ricapitolazione.

X

0. Voci correlate

- Firma Digitale
- Certificati e Certification Authority

1. Autenticazione Subject Fisico

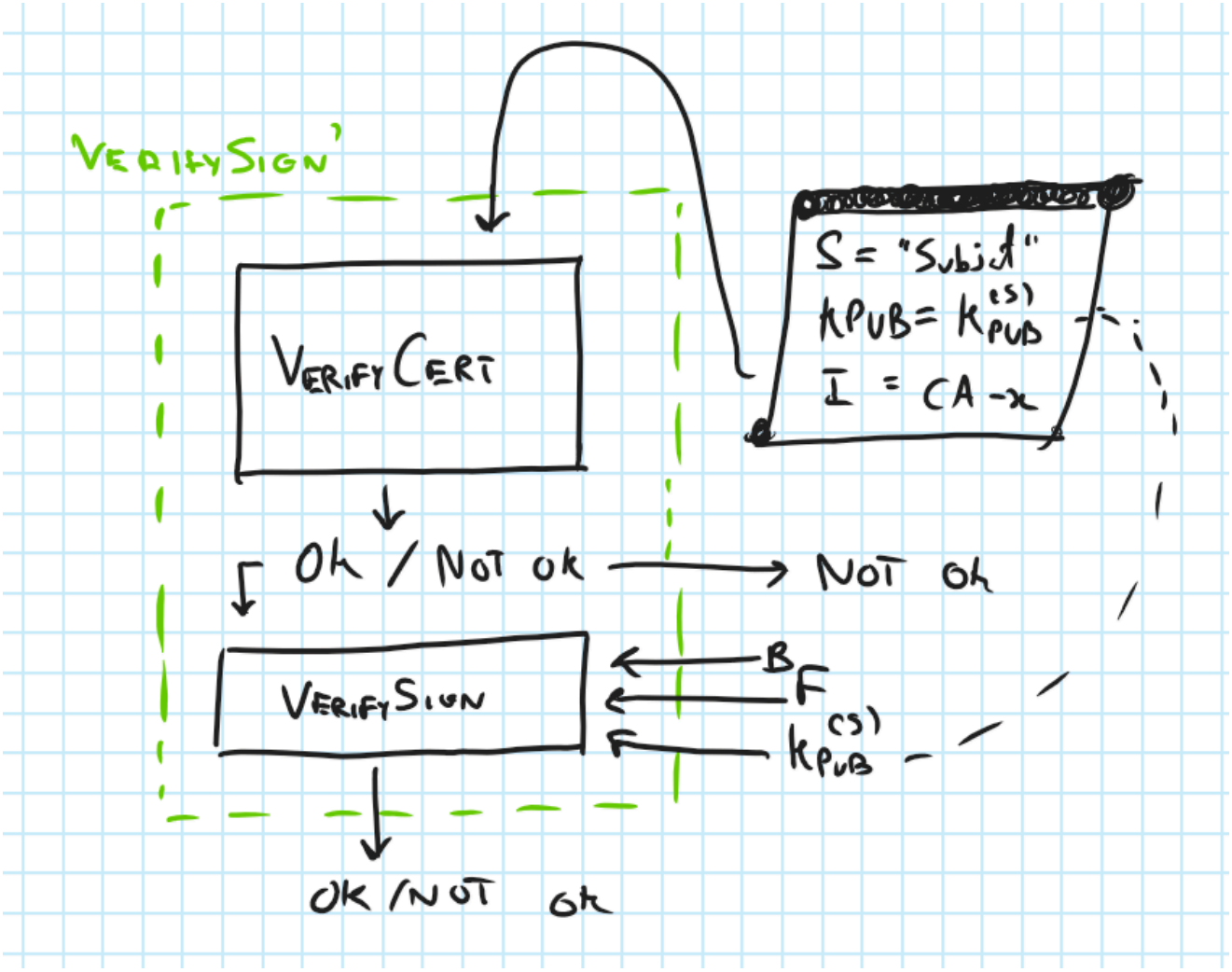
Vediamo la *firma digitale in pratica*.

Notiamo subito che bisogna richiedere anche l'*autenticazione del subject*, ovvero assicurarsi che un subject fisico abbia veramente diritto di utilizzare un certo subject.

In questo caso useremo i *certificati*, ovvero oltre ad inviare una firma per assicurare l'*integrità* ed *autenticazione* bisogna anche inviare il *certificato*.

Quindi in realtà la *verifica* delle firme digitali avviene come segue:

- Verifica il *certificato*, quindi assicurarsi che l'issuer del certificato sia nel *Trust set* e nel *Key set* e usare la *"bacchetta magica"* per verificare che sia effettivamente autentico ed integro
- Verificare la firma con la *chiave pubblica* nel certificato



Osservazione. Questo approccio ci ha appena risolto uno dei problemi di prima, ovvero la necessaria ipotesi della distribuzione di chiavi pubbliche. Usando i certificati, oltre a garantire l'associazione Subject Fisico e Subject, siamo anche in grado di specificare la *chiave pubblica*.

Osservazione. Se il certificato è valido e la verifica della firma ha successo, allora vuol dire che la firma F costruita da S è veramente costruita da S, non è stata modificata e S è veramente chi penso che sia

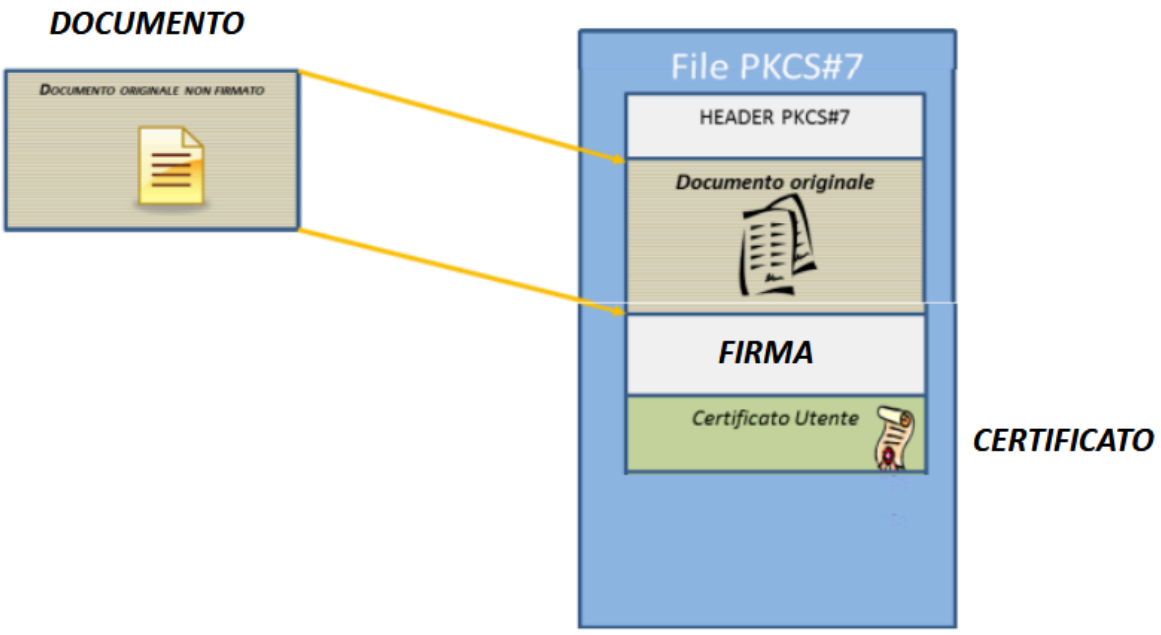
Se invece il *certificato non è valido* ma la *firma è comunque verificata*, vuol dire che F è costruita da S e non è mai stata modificata, tuttavia non posso garantire chi sia veramente S! Quindi non mi fido dell'associazione tra subject fisico e subject...

X

2. Modalità di Utilizzo Pratiche

Vediamo le principali modalità di utilizzo pratiche della *firma digitale*.

Innanzitutto è comune che il *documento* B, la *firma digitale* F e il *certificato* `cert` siano concatenate in un'unica sequenza di byte per questioni di comodità. Saranno quindi necessari dei *standard* per descrivere la struttura della sequenza, come ad esempio *PKCS7* e *p7m*.



Questo avviene molto più comunemente dei *file PDF* firmati digitalmente.

Per quanto riguarda l'*apposizione* e la *verifica* delle firme digitali, si ha che essi o sono delle *funzionalità integrate in programmi non specifici* o essi sono funzionalità di *programmi specifici*.

Come esempio di *"programma non specifico"*, si ha *Word, Excel, PDF readers* e solitamente accettano il documento unico con documento, firma e certificato. Inoltre, deve leggere il *personal set* per apportare firme e leggere il *Key Set / Trust Set* per verificare le firme.

Invece, alcuni programmi come *openSSL* sono *"fatti apposta"* per creare e verificare le firme digitali, e hanno moltissime varianti per file con documento, file e certificato.

Osservazione. Facciamo un paio di osservazioni terminologiche nel linguaggio comune

- Dire *"firmare col certificato"* è errato, siccome in realtà si usa la *chiave privata* per apporre una firma digitale
- Dire *"acquistare/creare la firma digitale"* è altrettanto sbagliato, in quanto non esiste *"la"* firma digitale, in realtà si intende l'acquisto della *coppia delle chiavi assimetriche* e *certificato con validità legale* per costruire la firma digitale.

Osservazione. (praticamente) In tutti i paesi hanno norme sugli *algoritmi, CA, distribuzione/revoca delle chiavi* per dargli una *"validità legale"*, ovvero un file PDF con firma digitale diventa *"equivalente"* ad un documento cartaceo firmato. Naturalmente ci sono molti requisiti, e in Italia si prevede un albo delle *Certification Authority* con *"validità legale"*. Non approfondiremo questo discorso.

X

3. Applicazione delle Firme Digitali

Le firme digitali ritrovano applicazione in *moltissimi ambiti*.

- Come primissimo esempio le firme digitali sono utili per le *applicazioni su documenti con validità legale*, come referti medici o verbali degli esami.
- Anche importante per gli *software eseguibili*, infatti un *software* creato da uno sviluppatore effettua un *"salto temporale/spaziale"* lunghissimo! Immagina quanti router attraversa il software...
  Pertanto è importante accertarsi che un'app scaricata sia sviluppata veramente dal developer del software. Nella pratica, vengono firmati o *dal sviluppatore software* o dal *gestore dello "app marketplace"*
  - Un sottoesempio più specifico è l'aggiornamento dei *sistemi operativi*; immagina cosa potrebbe fare un *network attacker* che riesce ad operare sui *"server di passaggio"*!

X

---

4. Considerazioni sulle Proprietà di Sicurezza

Facciamo delle ulteriori considerazioni sulle proprietà di sicurezza garantite dalla *firma digitale*.

L'*autenticazione* e *integrità* non implicano nessuna nozione sulla *"verità"* del contenuto di B.
Per convincerci di questo, facciamo un paio di esempi:

- La stringa *"Gli asini volano"* è certamente un'affermazione falsa, ma può essere comunque *firmata digitalmente* con chiave privata e verificata con chiave pubblica
- Ancora più assurdamente, una delle stringhe tra *"Dio esiste"* e *"Dio non esiste"* è vera ma entrambe possono essere comunque *firmate digitalmente*

Di conseguenza, applicando alle applicazioni pratiche abbiamo che:

- I documenti con *validità legale* non sono garantiti ad essere completamente veritieri
- I software con firma digitale possono avere *malware*, *backdoor* o *bug* per più motivi. Ragionando in negativo, un software *non firmato* può essere begigno e bug free.

Inoltre le proprietà di *autenticazione ed integrità* non bastano per alcune applicazioni.
Facciamo un esempio importante:

Esempio. Un file PDF è firmato e generato allo stesso istante di tempo T. Tuttavia, con la sola firma digitale è impossibile garantire che il file PDF sia veramente firmato nell'istante di tempo T, siccome la *"timestamp"* viene salvata solo nel contenuto B.

Pertanto richiediamo ulteriori *proprietà di sicurezza*, nel nostro caso possiamo risolvere istituendo le *"autorità temporali"* che generano delle *marche temporali firmati*.

X

---

5. TLS e Firma Digitale

Supponiamo cheun browser preleva un documento D senza firma digitale con HTTPs. Quanti messaggi HTTPs sono firmati? Nessuno, ovviamente.

E invece se il documento fosse firmato? Anche qui, dire che i messaggi HTTPs sono firmati è un *errore grave* siccome non sono firmati in nessun modo. I certificati vengono usati in un'altra maniera, certamente non per apporre/verificare firme digitali!

Recap Security

X

Esempi di recap sulla network security.

X

0. Voci correlate

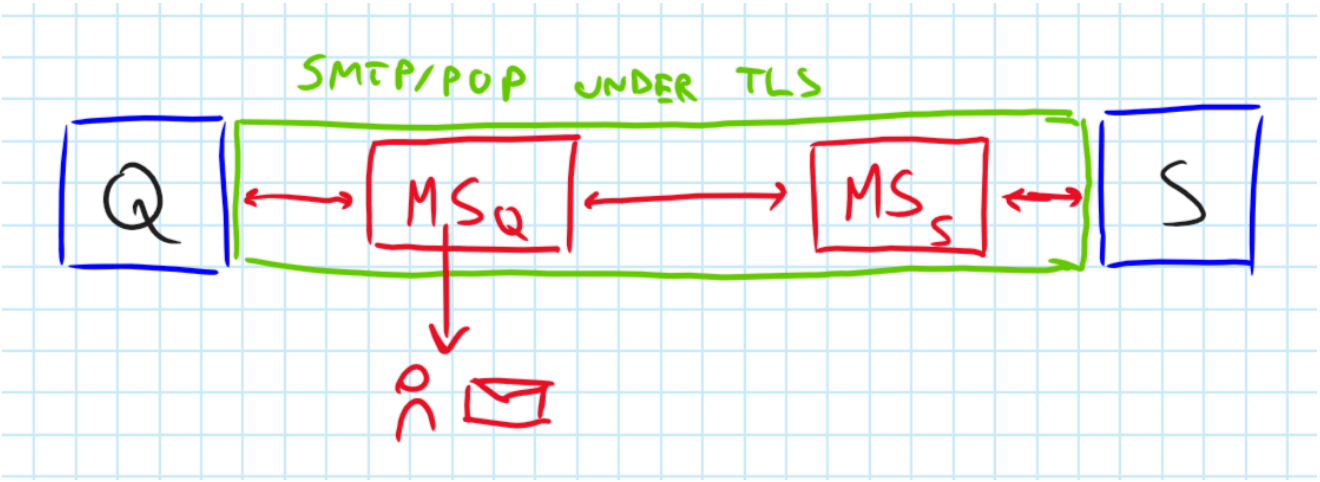
- Introduzione alla Network Security
- Firma Digitale
- Certificati e Certification Authority
- Crittografia a Chiave Pubblica
- Crittografia a Chiave Privata e HMAC

1. Recap Crittografia a Chiave Pubblica e Privata

Adesso facciamo un paio di esempi di ricapitolazione, per avere un modello mentale sulla *network security*.

Esempio. Supponiamo che la *stazione di polizia* comunichi con la *questura* via *mail*. Supponiamo che il *network attacker* operi solo sul canale di comunicazione TCP; allora per garantire le proprietà di sicurezza, è sufficiente usare applicare *TLS* su *SMTP/POP*.

Se invece il network attacker operasse anche sui *mail server "intermedi"*? In questo caso le difese garantite da TLS crollerebbero, siccome per il network attacker è sufficiente sfruttare i *mail server* per osservare/modificare le mail inviate/ricevute.



In questo caso bisogna applicare un ulteriore strato di sicurezza aggiuntivo. Un esempio è come segue:

1. Questura e stazione di polizia si concordano una *chiave privata* tra loro (o usando *certificati validi* o *usando un canale di comunicazione sicuro aggiuntivo*)

2. Crittare le mail con la *chiave simmetrica* e applicare l'*HMAC*

Notiamo che è sufficiente che *solo uno dei due lati* abbia la coppia chiave privata-pubblica. Anche se entrambi avessero la coppia delle chiavi private-pubbliche, sarebbe comunque consigliabile eseguire il procedimento di prima invece di comunicare direttamente con le coppie di chiavi in quanto sarebbe troppo costoso e mancherebbe anche l'HMAC, che richiede una *chiave simmetrica*.

Supponiamo che adesso il network attacker diventi un *node attacker* e può controllare i calcolatori alle due estremità. Allora anche in questo caso le difese crollerebbero e purtroppo non può essere risolta con *solo crittografia*, infatti il *node attacker* è in grado di leggere le chiavi simmetriche sui nodi, rendendola inutile.

Osservazione. Network attacker con *"poteri"* diversi implica meccanismi di sicurezza diversi. Ciò implica che quindi non possiamo essere certi su come si comporta l'attaccante e dobbiamo fare delle ipotesi *"sensate"*, infatti:

- Se *"sovrastimiamo"*, la realizzazione delle proprietà di sicurezza diventa molto costosa
- Se *"sottovalutiamo"*, allora il network attacker è in grado di penetrare le difese

Nel corso assumiamo che il network attacker sia in grado *solo* di operare sul canale di comunicazione vulnerabile.

X

---

2. Recap Firma Digitale

Consideriamo un altro esempio.

Esempio. Adesso consideriamo un nuovo scenario. La *stazione di polizia* vuole inviare un *file audio* alla *questura* con garanzia di *autenticazione ed integrità*.

Si firma il file audio, e per farlo bisogna supporre che la *stazione di polizia* abbia la *coppia di chiave pubblica-privata* e un *certificato valido*.

Quindi la stazione di polizia *P* invia alla questura *Q* la *firma*, il *file audio* e il *certificato valido* con chiave pubblica. Quando Q riceve tutto, esegui gli seguenti step:

1. Verifica il certificato, ovvero se l'issuer sta nel Trust Set, Key Set e nel caso positivo verifica l'autenticità e l'integrità del certificato
2. Controlla se la firma è valido o meno

Osservazione. La chiave pubblica della *Certification Authority* non viene usata nello secondo step, infatti serve solo per verificare i *certificati*.

X

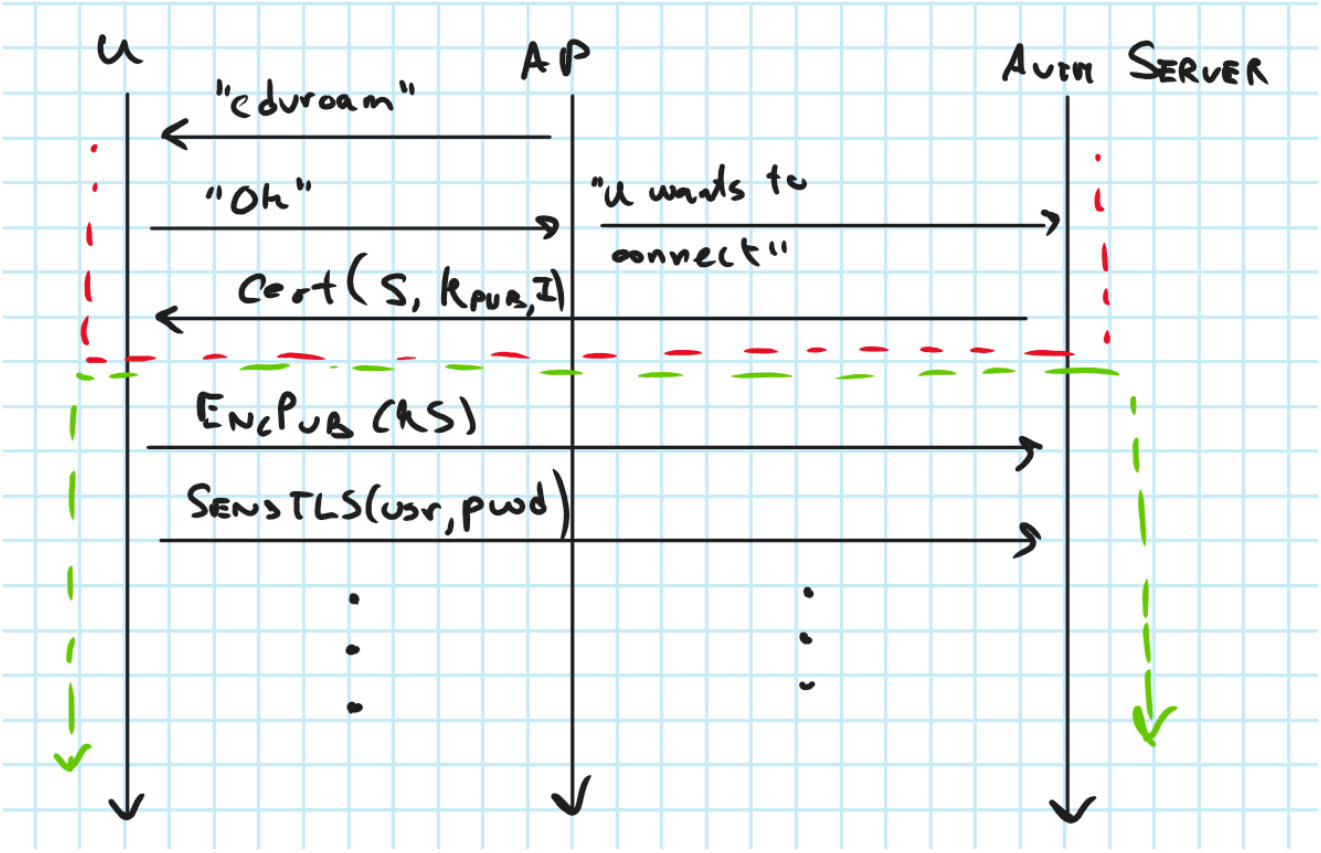
---

3. Curiosità Eduroam

Facciamo un ultimo esempio, come *curiosità*.

Esempio. Prendiamo uno scenario *"reale"*. Un utente vuole collegarsi a *eduroam* mediante un'access point dell'università; quello che succede è che:

1. L'access point invia messaggio *"eduroam"* all'utente per dire che deve collegarsi
2. L'utente si collega, e l'access point segnala al *server di autenticazione* che l'utente vuole collegarsi
3. Ovviamente la coppia username-password non può viaggiare in chiaro nella comunicazione tra *user* e *Auth server*; quindi dovranno concordare una chiave simmetrica tra loro per garantire le proprietà di sicurezza nella comunicazione
4. L'*auth server* invia un *certificato valido* all'utente per dire che è il server di autenticazione a cui mandare le credenziali
5. L'utente lo accetta e quindi concorderà la chiave simmetrica da cui comunica con le funzionalità TLS



Osservazione. In mancanza di configurazione, l'utente non può sapere chi è il *subject* (che certamente non è eduroam)! Infatti, il seguente attacco è possibile:

1. Il Network Attacker *"funge"* da Acess Point di eduroam
2. Invia un certificato valido con *subject "comprato"* (quindi che non sia veramente dell'Auth server)
3. L'user dovrebbe verificare il subject, ma essendo che non lo conosce (in quanto manca configurazione) lo accetta lo stesso

4. L'user comunica col network attacker

Morale della favola, è importante anche *configurare correttamente* eduroam, per evitare attacchi del genere.

Validazione dei Certificati

X

Validazione dei certificati "senza bacchetta magica": implementazione. Trust Set e Key Set in pratica.

X

0. Voci correlate

- Distribuzione e (cenni) Validazione dei Certificati
- Certificati e Certification Authority
- Crittografia a Chiave Pubblica in Pratica
- Firma Digitale

1. Validazione dei Certificati

Richiami. Ricordiamo che dato un *certificato* con la tripla *subject, chiave pubblica e issuer (certification authority* il software esegue gli seguenti passaggi per verificare la *validità, integrità e autorizzazione del certificato*.

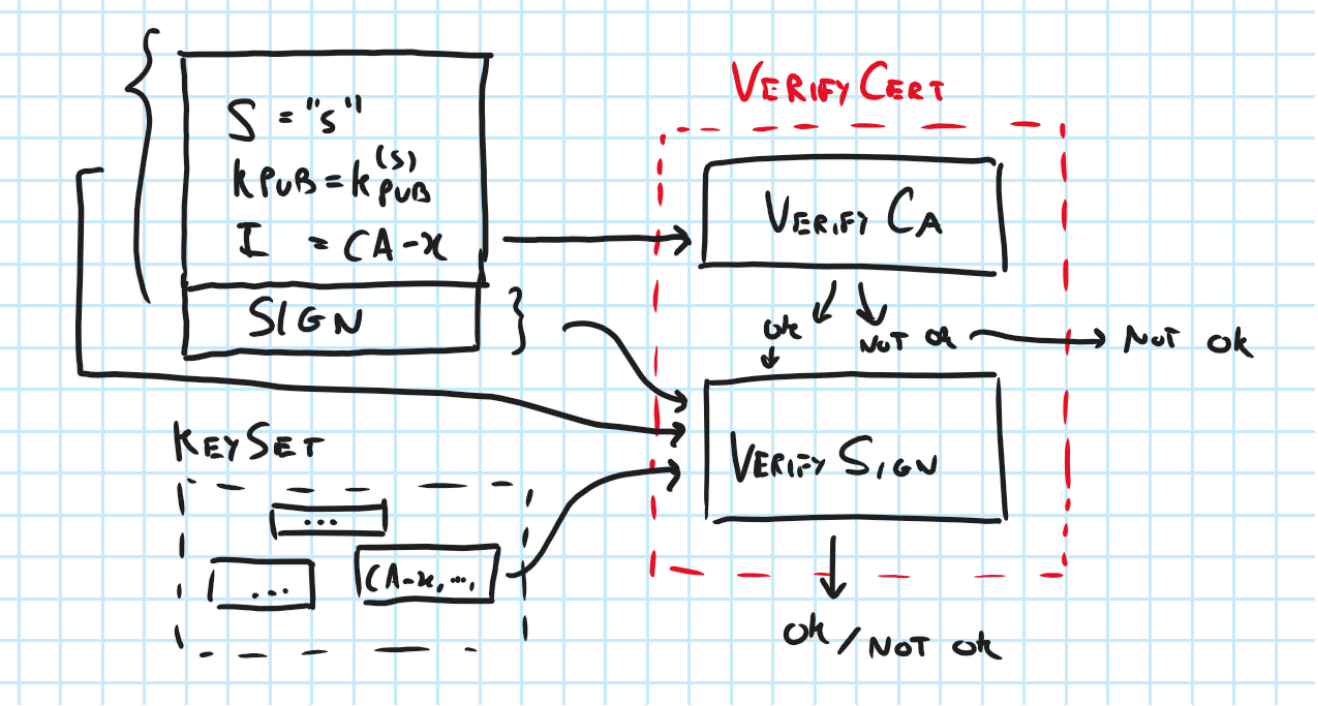
1. L'issuer appartiene al *Trust Set* o al *Key Set*? Se sì, continuare; altrimenti smettere
2. Verificare che il certificato sia *autentico ed integro* con uno *"strumento magico"* (!)

Adesso vediamo come realizzare lo step 2, ovvero verificare che un certificato sia *autentico ed integro*. Per farlo, basta applicare un ulteriore passaggio sui *certificati*: bisogna apporre una *firma digitale* e verificarla.

Tuttavia questo approccio comporta in sè un problema: per verificare la firma bisogna avere una *chiave pubblica*, da dove viene? La risposta sta nel *Key Set*, che era l'insieme che avevamo descritto come *"l'insieme delle CA su cui possiamo eseguire il test magico"*.

Con l'ottica delle firme digitali, adesso possiamo capire bene *come* rendano possibile il test di integrità e autenticazione.

Riassumiamo adesso il *"vero procedimento"* col seguente diagramma.



Osservazione. *"CA sta nel Key Set"* $\iff$ *"Ho la chiave pubblica di CA?"*

Osservazione. Per *validare* certificati uso solo il *test set e key set*, invece per *verificare firme* uso B e *chiave pubblica*, ma non c'entra il *Key Set*! Infatti, Il Key Set si usa solo per verificare il certificato da cui traggo la *chiave pubblica*. Quindi, dopo la verifica del certificato non uso più *Key Set*.

X

---

2. Trust Set e Key Set in Pratica

Facciamo un paio di *considerazioni pratiche finali* sul *trust set* e sul *key set*.

2.1. Aspetti Strategici del Trust Set

Q. Cosa succede se un *certification authority* CA-x generasse *avvolte* dei certificati falsi?

Vorrebbe dire che non ho più nessuna garanzia della verita dei suoi certificati e quindi delle associazioni chiavi pubbliche-subject.

Pertanto, *inserire CA-x* in *Trust Set* vuol diare avere *un potere enorme sui calcolatori*! Ogni affermazione presa da CA-x viene presa per versa...

Se invece inseriamo *CA-x* in più *Trust Set*, allora ha ancora un potere più grande.

Facciamo alcuni esempi che sottolineano proprio la *"potenza strategica"* delle CA in Trust Set.

Esempio. In Kazakistan ogni nodo è obbligato ad avere un *certificato del governo in Trust Set*, ciò vuol dire che il governo può intercettare ogni comunicazione e *"impersonare"* chiunque [2]

Esempio. A causa della guerra in Ucraina (o *"operazione militare speciale"*, a interpretazione libera...) e quindi delle sanzioni imposte, le *organizzazioni in Russia* non riescono più a pagare il rinnovo dei certificati. Come soluzione, il governo russo ha proposto di creare un *certificato nazionale* che vada a *"sostituire"* i certificati necessari. Notiamo che questa manovra funziona solo per *"nodi all'interno in Russia"*, invece all'esterno rimangono *"inacessibili"*. [3]

Esempio. Nel 2023 l'Unione Europea aveva proposto una legge per cui ogni *governo in UE* può avere un certificato per tutti i browser che operano nel territorio UE. Questo ha attratto molti criticismi, come ad esempio da parte di Mozilla. [4]

Q. Come possiamo sapere che una CA-x in Trust Set dica sempre la verità?

Non si può, è un'assunzione data per scontata. Notare che tutto il sistema della crittografia asimmetrica si regge su questo presupposto "*irrealistico*".

2.2. Trust Set e Key Set in Pratica

Concludiamo il discorso considerando il *trust set* e *key set* in pratica.

Nella crittografia, ci sono tre *insiemi fondamentali*:

- Personal Set
- Trust Set
- Key Set

In astratto, sono degli insiemi *"unici"* e *"separati"*; tuttavia, nella realtà questi insiemi potrebbero:

- Avere nomi diversi
- Alcuni file possono contenere un *"mescolamento"* di elementi di Personal Set, Trust Set, Key Set, quindi gli insiemi possonoe ssere mescolati tra loro
- Essere specificati con formati diversi

Solitamente, si trovano in *files*.

Esempio. Su Windows c'è una certella predefinita e viene usata dalle *"applicazioni del sistema operativo"*. Invece, le applicazioni *"esterne"* hanno solitamente (ad oggi) una loro cartella dedicata.

Q. Come bisogna proteggere gli insiemi Key Set / Trust Set?

Naturalmente li proteggiamo *solo* in scrittura, invece non serve proteggerli *in lettura* (anzi, non bisogna!) siccome dovranno essere letti dai software.

Q. Come arrivano i certificati? Ovvero, da dove vengono?

Arrivano *già installati nei sistem ioperativi* o durante la fase dell'*installazione del programma*. Quindi questa fase è *molto critica!* Infatti, stiamo decidendo chi mettere nel Key Set ma sopratutto nel Trust Set!!!

Ognugno può generare certificati autofirmati e metterli nel Trust Set / Key Set, quindi non avrebbe senso verificare i *certificati self-signed*.

Pertanto, bisogna assumere che ci sia un *canale di comunicazione aggiuntivo sicuro* tra il *fornitore del S.O.* e il *S.O.* stesso.

Per concludere, ripetiamo la citazione di R. Needham sulla crittografia:
"Whenever anyone says that a problem is easily solved by cryptography, it shows that he doesn't understand it"

X

---

1. https://research.mozilla.org/files/2025/03/the_state_of_https_adoption_on_the_web.pdf ↩
2. https://www.itpro.com/network-internet/34051/kazakh-government-will-intercept-the-nation-s-https-traffic ↩
3. https://medium.com/coinmonks/russia-establishes-its-own-tls-certificate-authority-to-avoid-sanctions-a8221b72b729 ↩
4. https://last-chance-for-eidas.org/ ↩