Co-funded by the Erasmus+ Programme of the European Union

Modulo di istruzione di InCyT

Information Security for Employees **Unità 2 Settimana 3** **Nome dell'unità: Introduzione alla crittografia**	
Attività di apprendimento	❑ **Webinar** ❑ **Exercises** ❑ **Workshop** ❑ **Presentation** ❑ **Other**
Responsabile dell'apprendimento	
Obiettivi dell'attività di apprendimento	1. Crittografia 2. Crittoanalisi 3. Algoritmo One Time Pad 4. Algoritmi simmetrici 5. Algoritmi asimmetrici 6. Funzioni hash 7. Firme e infrastrutture a chiave pubblica 8. Token crittografici 9. Meccanismo challenge-response
Competenze da acquisire	• Competenze di base sugli algoritmi crittogrrafici e le infrastrutture • Technical Skill 2 - TS2
Durata dell'attività di apprendimento	1 week
Requisiti di apprendimento	***Cosa è richiesto*** • Non è richiesta alcuna conoscenza specifica precedente

Informazioni sintetiche sul modulo

La crittografia riguarda le metodologie per nascondere le informazioni dalla divulgazione, sia durante una comunicazione che per scopi di archiviazione. I suoi obiettivi sono raggiunti con la progettazione e l'implementazione di protocolli che impediscono a terzi non autorizzati di recuperare le informazioni. In una comunicazione protetta il terzo avversario non può accedere alle informazioni trasmesse, in caso di archiviazione il terzo non può risalire al contenuto di quanto archiviato.
In crittografia, la terza parte è sempre un'entità dannosa che mira a recuperare informazioni a cui non deve avere accesso, sia per motivi di privacy, segreti commerciali o sicurezza nazionale. La riservatezza e l'integrità dei dati, l'autenticazione delle parti coinvolte e il non ripudio di ciò che la fonte dei dati ha trasmesso o archiviato sono i principi fondamentali della moderna crittografia.
Dopo una parte introduttiva, in cui vengono descritti i concetti fondamentali, sono presentate le tecniche crittografiche di base, partendo dall'algoritmo One Time Pad per poi considerare le due famiglie crittografiche di algoritmi: algoritmi simmetrici e algoritmi asimmetrici (o “a chiave pubblica”), per poi farne un confronto. Viene anche discusso il ruolo delle funzioni hash come introduzione alle firme digitali. Viene presentata in profondità una descrizione del concetto chiave di Infrastrutture a chiave pubblica (PKI) con alcuni dettagli sui meccanismi e formati standard. Per concludere, vengono fornite alcune informazioni sui token crittografici e sul meccanismo di sfida (challenge-response).

Contenuto del materiale didattico

Introduzione

La **crittografia** è il meccanismo mediante il quale il contenuto di un messaggio viene nascosto ad altre entità.
In altre parole la crittografia si occupa di cifrare i messaggi, cioè prendere un testo in chiaro, utilizzare un algoritmo e in genere una chiave e rendere incomprensibile il contenuto del messaggio.
La **crittoanalisi** studia come decifrare un messaggio senza conoscere la chiave. Crittografi e crittoanalisti sono in realtà le stesse persone: chi scrive gli algoritmi crittografici li analizza anche per trovarne i punti deboli o convalidarne la robustezza. Quando si ha a che fare con la sicurezza, l'idea di poter affrontare i problemi di protezione senza sapere come funzionano le tecniche di attacco non è mai molto efficace.
Le informazioni nei messaggi sono generalmente ridondanti: i messaggi utilizzano generalmente un numero di bit molto più elevato di quello realmente necessario per trasportare le informazioni. La

ridondanza delle informazioni facilita la crittoanalisi: significa che i messaggi possono mostrare una struttura che rende più facile identificare il contenuto del messaggio, poiché non tutti i messaggi sono ugualmente probabili. Se pensiamo ad un testo rappresentato con normali caratteri ASCII in byte, sappiamo che le combinazioni di bit che possono rappresentare messaggi significativi sono relativamente poche rispetto a tutte le possibili combinazioni. Ossia i messaggi significativi che possono essere rappresentati con un certo numero di byte sono pochissimi, gli altri sono sequenze sconclusionate di byte. Ciò consente di identificare i messaggi validi eliminando quelli che non contengono informazioni vere. Tutto ciò aiuta enormemente l'attività del crittoanalista, quindi lo scopo dell'algoritmo di crittografia è anche quello di ridurre questa ridondanza per nascondere meglio le informazioni.

Esempio:
La codifica di una sequenza dei quattro colori BLU, VERDE, ROSSO e GIALLO.
Questi possono essere codificati:

- con 2 bit (00, 01, 10, 11)
- come 4 diverse lettere maiuscole
- cifrando i nomi dei colori

Nel primo caso viene utilizzato il numero minimo di entità (bit), sostituendo ogni nome di colore con I numeri a 2 bit che si è stabilito di associare a ciascuno di essi, si nasconderà la sequenza originale, ma non l'ordine in quanto esiste una corrispondenza tra COLORE e il numero a 2 bit. Nel secondo caso, poiché ogni lettera utilizza 8 bit nel codice ASCII, 6 bit sono ridondanti per ogni colore→75% di ridondanza e otteniamo lo stesso grado di “occultamento” che hanno i 2 bit da soli.
Si consideri il seguente cifrario molto semplice, il cosiddetto cifrario di Cesare, che non fa altro che scorrere le lettere dell'alfabeto di un certo numero di posizioni. Ad esempio, se li facciamo scorrere di 3, tutte le "A" diventano "D", tutte le "B" diventano "E" e così via. Questo è un codice molto semplice, attribuito a Giulio Cesare. Se i nomi dei colori sono crittografati con questo metodo, lo schema ripetuto delle loro lettere codificate è facilmente identificabile (il codice ASCII di R seguito dal codice ASCII di E seguito dal codice ASCII di D sono facilmente riconoscibili come entità). Quindi la sequenza può essere individuate comunque, se poi le parole sono note, basta contare il numero di lettere per identificare le parole originali.
Questo semplice esempio ci permette di capire che la crittoanalisi non è un'analisi indipendente dal contenuto del messaggio: un messaggio completamente reso casuale sarebbe molto più difficile da identificare e analizzare rispetto a un messaggio che ha una struttura riconoscibile.
Un algoritmo di crittografia nasconde anche le informazioni ridondanti, quindi l'applicazione di un algoritmo di compressione a un messaggio crittografato non porta molti vantaggi, perché ridurrebbe il tasso di compressione. Se vogliamo usare la compressione e la crittografia, dobbiamo prima usare la compressione e poi la crittografia.

Alcuni tipi di attacchi crittografici e algoritmi si basano sulla conoscenza di parte del testo. In alcuni casi, conoscere parte del testo consente di risalire alla chiave e quindi accedere al resto del testo. Questo è molto importante quando si parla di comunicazioni di rete, perché in queste i protocolli ai diversi livelli di solito hanno una certa quantità di informazioni comunque riconoscibili, ad esempio in un pacchetto IP sappiamo che l'intestazione contiene l'indirizzo del mittente e il indirizzo del destinatario.

L'algoritmo One Time Pad

L'algoritmo One Time Pad (OTP) è molto semplice concettualmente e da implementare e ha una caratteristica che lo rende particolarmente interessante: è dimostrato che senza conoscere la chiave è impossibile risalire al testo originale, per questo viene considerate l'algoritmo "perfetto". Da quanto segue si evnicerà perché, pur avendo trovato un algoritmo così buono, è necessario utilizzarne altri.

L'OR ESCLUSIVO ⍰ (XOR) è una funzione logica che dà risultato VERO (True) solo quando i due operandi sono diversi. Opera su valori logici, ma tradotto in cifre binarie (solitamente True=1 e Falso=0) la tavola di verità è la seguente:

$0 \oplus 0 = 0$

$0 \oplus 1 = 1$

$1 \oplus 0 = 1$

$1 \oplus 1 = 0$

Da questa tabella risulta chiaro che se $X \oplus Y \rightarrow Z$, then $Z \oplus Y \rightarrow X$ and $Z \oplus X \rightarrow Y$. L'OR ESCLUSIVO è una funzione molto semplice e facile da implementare anche come circuito elettronico.

Il meccanismo OTP viene utilizzato per trasferire un messaggio da due parti. Nella letteratura scientifica è prassi comune identificare gli attori di una trasmissione con nomi propri, di solito vengono chiamati Alice (A) e Bob (B). Quindi verrà descritto come Alice può trasmettere in modo sicuro un messaggio (*ClearText* – Testo in chiaro) di *n* byte a Bob.

L'algoritmo One Time Pad richiede una fase di configurazione:

- Viene generata una sequenza casuale di esattamente *n* byte, questa è il l'One Tima Pad o semplicemente la *Chiave*;
- L'OTP viene precedentemente condiviso in modo sicuro tra le due parti (fornito in anticipo, inviato tramite un canale già protetto, ecc.).

Quando entrambe le parti possiedono la chiave, possono utilizzarla per trasmettere messaggi. Nel caso Alice volesse mandare un messaggio a Bob:

- Alice calcola l'OR ESCLUSIVO byte per byte del *ClearText* con il *Chiave*, producendo il *CypherText (Testo cifrato)*:

Cifratura: *ClearText* ⊕ *Chiave* → *CypherText*

- Il risultato *CypherText* viene trasmesso da Alice a Bob su un canale potenzialmente non sicuro
- Bob calcola byte per byte l'OR ESCLUSIVO del *CypherText* con la *Chiave*, recuperando così l'originale *ClearText*:

Decifratura : *CypherText* ⊕ *Chiave*→ *ClearText*

Esempio

Supponiamo che sia Alice che Bob abbiano già concordato una OTP casuale (*Chiave*).

Alice vuole inviare "HELLO" a Bob (nel codice ASCII, gli spazi sono qui introdotti per facilitarne la lettura):

	H	**E**	**L**	**L**	**O**
PlainText:	01001000	01000101	01001100	01001100	01001111
			⊕		
Key:	10101001	01110010	10110010	10000110	11000110
			=		
CypherText:	11100001	00110111	11111110	11001010	10001001

Bob riceve il *CypherText* and applica *Key*:

CypherText:	11100001	00110111	11111110	11001010	10001001
			⊕		
Key:	10101001	01110010	10110010	10000110	11000110
			=		
PlainText:	01001000	01000101	01001100	01001100	01001111
	H	**E**	**L**	**L**	**O**

Il *PlainText* è stato recuperate correttamente.

Essendo la *Chiave* una sequenza casuale, non c'è modo di dire se un certo bit della sequenza sia '0' o '1', poiché sono ugualmente probabili. Senza la *Chiave*, dato un bit del *CypherText*, non abbiamo alcuna possibilità di risalire a quale potrebbe essere stato il valore del bit originale. Questo vale per i singoli bit, per l'intero messaggio, per ogni frammento del nostro messaggio.
Nel caso una parte della *Chiave* sia nota, è possibile recuperare la parte corrispondente del *ClearText*, ma senza il resto della *Chiave* non ci sono informazioni sugli altri bit del *ClearText*. Pertanto, la crittoanalisi menzionata precedentemente non è efficace. In questo caso, per scoprire il resto del messaggio originale, la parte rimanente della *Chiave* è indispensabile.

Problemi del meccanismo One Time Pad

La forza del meccanismo OTP si basa sulla lunghezza della *Chiave,* questa è esattamente uguale alla lunghezza del testo ed è da utilizzarsi una sola volta. Lo svantaggio principale è che la trasmissione dell'OTP presenta gli stessi problemi del messaggio.
In alcuni casi questo problema è facilmente risolvibile: ad esempio a un'ambasciata potrebbe essere fornita una valigetta diplomatica contenente una chiave molto lunga, quindi i messaggi che verranno scambiati con tale ambasciata potranno essere crittografati consumando gradualmente parti della chiave. Sono usi molto particolari, ma piuttosto efficaci.
Se si considera la comunicazione in rete, i reali utilizzi dell'One Time Pad sono molto limitati. C'è bisogno di qualcosa di più flessibile, qualcosa che consenta lo scambio di una chiave che può essere utilizzata per molto tempo.

Consideriamo i vari problem dell'OTP.

Trasmissione offline preliminare

Le chiavi devono essere concordate in qualche modo dalle parti in gioco, questa necessità è un problema molto importante per l'uso efficace della crittografia che si aggiunge a quello della robustezza degli algoritmi e quindi al contrasto della crittoanalisi.
La gestione delle chiavi consiste nel fare in modo che tutti e soli i soggetti autorizzati abbiano accesso alle chiavi, quindi si tratta di scambiarle in anticipo e di assicurarsi che le chiavi scambiate siano le quelle corrette, essendo il numero di chiavi anche molto elevato.

Uso *una tantum*

L'One Time Pad viene utilizzato una sola volta, come suggerisce il nome, questo è una limitazione importante. Se si usasse la stessa chiave su più messaggi si potrebbe fare qualche analisi e la sicurezza verrebbe ridotta.

Generare numeri casuali in un sistema deterministico è molto difficile

I normali computer generano numeri *pseudocasuali,* ossia ogni numero viene calcolato dal precedente, questo produce una sequenza di numeri che necessariamente si ripetono, anche se dopo un periodo molto lungo. Questi non sono quindi *veri* numeri casuali in quanto un computer è una macchina deterministica. In crittografia, tuttavia, questa caratteristica è deleteria. Se è possibile in qualche modo ricavare un numero "casual" dal precedente, le protezioni offerte da algoritmi e protocolli crittografici diventano molto meno sicure.

Scrivere il codice che implementa algoritmi crittografici non è banale: vi è una grande quantità di dettagli a cui prestare attenzione, tanti problemi che sono stati individuati negli anni dagli sviluppatori del settore e che solo l'esperienza in questa specifica area permette di gestire correttamente. Ciò significa che non ci si improvvisa programmatori di codici crittografici né è opportune svilupparli internamente alla propria azienda senza la dovuta competenza, è opportune invece di utilizzare librerie ampiamente utilizzate e convalidate da esperti.

Algoritmi diversi da OTP

Oltre l'OTP si hanno algoritmi utilizzano chiavi a lunghezza fissa. Se si usa una chiave a n bit, si possono avere 2^n chiavi diverse, se il valore di n è abbastanza grande non è efficace utilizzare un attacco *brute force* (cioè provare tutte le chiavi possibili): ci vorrebbero così tanti anni per trovare la chiave giusta in modo che "probabilmente" il contenuto del messaggio non sarebbe più di interesse. Ad esempio, con uno dei supercomputer più veloci disponibili attualmente, il controllo di una chiave a 256 bit richiederebbe $3{,}67\times10^{55}$ anni. Chiaramente più è lunga la chiave e più è importante il costo computazionale anche per coloro che cifrano e decifrano legittimamente, ma l'impatto è limitato.

Gli algoritmi crittografici si dividono in simmetrici e asimmetrici.

Algoritmi simmetrici

La più tipica e antica famiglia di algoritmi crittografici, quelli a cui pensiamo di solito quando pensiamo alla crittografia dei messaggi, sono gli ***algoritmi simmetrici***, così chiamati per la simmetria delle operazioni di cifratura e decifratura. Questi algoritmi utilizzano *un'unica chiave* che deve essere nota sia al mittente sia al destinatario; questi la usano per cifrare un messaggio e quindi per decifrarlo.

Lo schema simmetrico è quello stesso dell'OTP, ma questi algoritmi usano una *Chiave* di lunghezza fissa (es. 256 bit) che viene utilizzata ripetutamente in EXOR con il testo (eventualmente dopo averla modifica in modo deterministico ogni volta):

- **Cifratura:**
  funzione_cifratura(ClearText, Chiave)→ CypherText
- **Decifratura:**
  funzione_decifratura(CypherText, Chiave)→ ClearText

Più in generale, anche se usiamo la stessa *Chiave* la funzione utilizzata per decifrare non è necessariamente la stessa utilizzata per cifrare. Ad esempio, se consideriamo il cifrario di Cesare, è chiaro che la funzione che decifra il messaggio fa scorrere l'alfabeto a sinistra e non a destra, ma la decifratura utilizza la stessa chiave: il numero 3.

Alcuni importanti algoritmi a chiave simmetrica

DES (Data Encryption Standard)

Questo algoritmo storico, ormai obsoleto, utilizza una chiave a 56 bit che, alla fine degli anni '70 (quando è stato sviluppato l'algoritmo), era assolutamente adeguata per molti usi: un attacco brute force su una chiave a 56 bit per decifrare il messaggio era allora impraticabile, usando i computer disponibili all'epoca e considerando i contesti in cui tale algoritmo veniva utilizzato. 40 anni dopo l'algoritmo DES non durerà che pochi minuti su un computer di casa. Questo è un aspetto molto importante: la lunghezza della chiave deve essere scelta in base alla *effettiva* capacità di calcolo disponibile.

Una chiave considerata abbastanza robusta ora potrebbe non essere più così forte tra 20 anni. In generale si tende ad utilizzare chiavi più lunghe del minimo indispensabile per garantire che ci sia un certo margine temporale, sia perché le capacità di calcolo aumentano, ma anche perché la disciplina della crittoanalisi fa progressi e potrebbe trovare punti deboli nell'algoritmo. Contrariamente a quanto si potrebbe pensare, normalmente un algoritmo crittografico non “si rompe” all'improvviso, quando viene scoperto un difetto in un algoritmo questo non passa immediatamente da sicuro a completamente insicuro e alla mercè di tutti.

I ricercatori analizzano l'algoritmo e possono eventualmente identificare *piccoli* punti deboli in grado di facilitare, in alcuni casi, un attacco. Altri potrebbero trovare qualche ottimizzazione dell'algoritmo per risolvere il problema. Con il passare del tempo l'algoritmo progressivamente si indebolisce a causa dei problemi irrisolvibili, fino a non essere più adeguato per determinati usi o per niente. L’erosione dell’efficacia crittografica è progressiva, questo lento indebolimento è un problema difficile da gestire perché non si può sapere quali progressi ci saranno in futuro dal punto di vista della crittoanalisi.

AES (Advanced Encryption Standard, noto anche con il nome originale Rijndael)

AES è stato scelto in base non solo alla robustezza, ma anche all'efficienza dell'implementazione in architetture tipiche. Non è stato cercato alcun algoritmo segreto perché un algoritmo segreto non è considerato più robusto di uno non segreto: al contrario, l'algoritmo doveva essere analizzato da

quanti più crittoanalisti possibile per garantire che fosse intrinsecamente robusto. Un algoritmo sicuro ma troppo lento, ossia troppo costoso in termini di risorse di calcolo, non sarebbe utilizzabile. Era chiaro che questo vincolo era considerate il più difficile da realizzare, rispetto a quello della robustezza.

Algoritmi asimmetrici (o a chiave pubblica)

Gli algoritmi asimmetrici non utilizzano una sola chiave, ma due: una chiave viene utilizzata per cifrare e l'altra per decifrare. Pertanto, nello scambio di messaggi, mittente e destinatario non utilizzano la stessa chiave.Queste chiavi sono generate e gestite in coppia: ciò che una chiave (una qualsiasi della coppia) cifra può essere decifrato solo dall'altra chiave della coppia.

Il motivo per utilizzare algoritmi asimmetrici derivano dai problemi che la crittografia simmetrica ha:

- Le comunicazioni riservate tra 2 entità richiedono che una chiave univoca sia condivisa solo tra loro, se le entità lo sono *n* allora sono necessarie $n\cdot(n-1)/2$ chiavi diverse per consentire una comunicazione riservata tra ciascuna coppia di parti
- Sia il mittente sia il destinatario conoscono la loro chiave tra loro condivisa, quindi il *non ripudio* (il ripudio è negare di essere l'autore di un certo messaggio) non è possibile

Il primo problema non riguarda tanto il numero di chiavi che ogni soggetto deve gestire, il problema è *condividerle*. Ogni soggetto deve concordare una chiave con ciascuno degli altri n-1 soggetti, cosa che in realtà è un'operazione complessa; questo rende la crittografia simmetrica poco pratica per diversi utilizzi.

Il *non-ripudio* consiste nel voler essere sicuri che un certo messaggio sia stato effettivamente inviato da un determinato soggetto. È chiaro che se un soggetto riceve un messaggio da qualche altro soggetto e i due soggetti sono gli unici a conoscere la chiave utilizzata per cifrarlo, ciascun soggetto è sicuro che l'altro soggetto sia l'autore del messaggio. Viceversa, ogni soggetto sa che solo l'altro soggetto può leggere il messaggio. Il problema si presenta quando è necessario dimostrarlo a terzi, in un contenzioso, dato che ciascuno è in grado di creare un messaggio, cifrarlo e quindi dire che è l'altra parte l'autore. Il problema del *non-ripudio* non è risolvibile utilizzando la crittografia simmetrica.

Lo schema asimmetrico è simile a quello simmetrico, ma le chiavi utilizzate sono le due che vengono generate in coppia:

- **Cifratura:** *funzione_cifratura*(*ClearText*,***Chiave1***)→ *CypherText*
- **Decifratura:** *funzione_decifratura*(*CypherText*,***Chiave2***)→ *ClearText*

I nomi ***Chiave1*** e ***Chiave2*** sono stati usati al posto di *Chiave di cifratura* e *Chiave di decifratura* perché le due chiavi sono operativamente intercambiabili. Il soggetto che genera le chiavi generalmente ne mantiene una privata/segreta (denominata *chiave privata*) e rende pubblica l'altra (denominata *chiave pubblica*), da cui il nome della classe di algoritmi che è anche chiamata *algoritmi a chiave pubblica*.

La funzione utilizzata per cifrare non è necessariamente la stessa utilizzata per decifrare. L'importante è che insieme costituiscano l'algoritmo crittografico, una parte si basa su una chiave e l'altra parte sull'altra chiave.

Le due chiavi hanno proprietà matematiche tali che trovare una chiave a partire dall'altra è decisamente complesso e richiede quantità di tempo estrememente grande. In particolare, anche conoscendo la chiave pubblica, il ClearText e il CypherText, la ricostruzione della chiave privata richiederebbe una quantità di tempo eccessiva.

Utilizzi dell'algoritmo asimmetrico

Si supponga (i dettagli verranno forniti in seguito) che la *chiave pubblica* di un soggetto sia disponibile pubblicamente e il suo proprietario sia l'unico a conoscere la chiave privata corrispondente. Essendo detta chiave pubblica a disposizione di chiunque, chiunque può utilizzarla per cifrare un messaggio, ma solo il proprietario della chiave privata corrispondente potrà decifrare il messaggio: questo fornisce **riservatezza** nella trasmissione del messaggio.

Si è passati da una situazione in cui un soggetto aveva bisogno *n* chiavi diverse per comunicare in modo riservato con *n* altri soggetti ad una situazione in cui è necessaria *una sola* chiave, la chiave pubblica del destinatario, affinché *tutti* possano inviare un messaggio riservato a quel destinatario.

Per consentire a *n* soggetti di comunicare tra loro, ogni soggetto deve avere una coppia di chiavi, quindi il numero totale di chiavi è $2 \cdot n$. È interessante notare non tanto che le chiavi siano diminuite di numero, ma che essendo chiavi pubbliche queste possano essere comunicate in un modo molto più semplice. Se qualcun altro viene a conoscenza di quelle chiavi pubbliche, non è affatto un problema. Pertanto, queste chiavi possono essere realmente pubblicate, messe a disposizione di tutti e tutti possono utilizzarle senza indebolire la protezione dei messaggi.

Se il soggetto è l'unico proprietario di una chiave privata (segreta), la cui controparte pubblica è disponibile a tutti, chiunque può utilizzarla per verificare (decodificare) i messaggi crittografati con la chiave *privata* del soggetto.

Se è possibile utilizzare la chiave pubblica di un soggetto per decifrare un messaggio, significa che solo il soggetto che possiede la chiave privata corrispondente può aver cifrato il messaggio, questo garantisce il **non-ripudio** del messaggio.

C'è ancora un problema da risolvere: fare in modo che ogni chiave pubblica sia realmente associata al legittimo soggetto. È necessario un metodo di certificazione super partes, questo è fornito da un'*Infrastruttura a chiave pubblica* (PKI), discussa più avanti.

Per inciso, gli algoritmi asimmetrici possono essere utilizzati anche come *algoritmi challenge-response* (sfida-risposta). Affinché il soggetto da A possa *autenticare* B (verificare che B è veramente chi afferma di essere), A invia (in chiaro) un numero casuale a B; quindi B lo crittografa con la sua chiave privata e lo rimanda ad A. Se A riesce a decifrare il messaggio con la chiave pubblica di B, A è sicuro che l'altra parte sia B, in quanto è l'unico in grado di cifrare qualcosa che la chiave pubblica di B può decifrare.

Vantaggi e svantaggi

Il numero totale di chiavi è notevolmente diminuito, il che diminuisce la complessità di garantire la corretta gestione di tutte le chiavi (e solo il proprietario conosce la chiave segreta). Poiché le chiavi pubbliche sono messe a disposizione del pubblico, non sono necessari accordi specifici tra soggetti diversi. Questo uso delle chiavi pubbliche è, ad esempio, molto importante nell'e-commerce, perché l'e-commerce non può richiedere un accordo a priori del venditore con i clienti.

Il legame tra le due chiavi che, allo stesso tempo, garantisce che una chiave non può essere calcolata dall'altra e che ciò che è crittografato con una viene decifrato con l'altra, rende gli algoritmi asimmetrici molto più complessi in termini di problemi matematici rispetto a quelli simmetrici. Questi problemi matematici richiedono essenzialmente algoritmi più lenti rispetto agli algoritmi simmetrici e l'uso di chiavi più lunghe, almeno con gli algoritmi attualmente utilizzati. Pertanto, se da un lato c'è una maggiore flessibilità, dall'altro sono più onerosi in termini di risorse di calcolo.

Alcuni importanti algoritmi asimmetrici

RSA*(dalle iniziali di Ronald* ***R****ivest, Adi* ***S****hamir e Leonard* ***A****dleman)*

Questo algoritmo si basa sulla complessità della fattorizzazione dei primi: è complesso/intrattabile trovare i due numeri primi dal risultato, è un problema difficile. Si sa anche che non è facile trovare numeri primi molto alti, in quanto non c'è un metodo che permetta di scoprire numeri primi se non provare a fattorizzarli e scoprire se sono primi. Ciò significa che lavorare su numeri molto grandi, scoprire quali sono i due numeri primi da cui un dato numero è derivato, è un problema che richiede notevoli risorse di calcolo. D'altra parte, le operazioni legittime possono essere eseguite con risorse informatiche limitate.

Crittografia ellittica *(ECC - Crittografia a curve ellittiche)*

Il “problema difficile” in questo caso è il calcolo del logaritmo discreto (in un campo finito). Un "campo finito" significa che i valori interi sono in un insieme limitato (definizione molto approssimativa).

Il problema ECC è più complesso della fattorizzazione dei numeri primi, quindi si ha la stessa sicurezza con chiavi più corte. Gli algoritmi ECC sono più veloci dell’RSA.

Confronto di algoritmi simmetrici e asimmetrici

Gli algoritmi asimmetrici sfruttano problemi matematici diversi da quelli della crittografia simmetrica. Ciò significa che le lunghezze delle chiavi non sono paragonabili a quelle della crittografia simmetrica. Il problema con la crittografia simmetrica è che la chiave sia abbastanza lunga da impedire di provare tutti i numeri possibili che potrebbero corrispondere a una chiave, quindi con una chiave a 128 bit ci sono 2^{128}(ovvero $3{,}4 \cdot 10^{38}$) possibili chiavi. Gli algoritmi simmetrici attuali usano chiavi a 128, a 256 e a 512 bit.

Se consideriamo ad esempio RSA, che sfrutta i numeri primi, è chiaro che non tutte le chiavi sono possibili, perché legate alla presenza di numeri primi, non tutti i numeri sono primi e quindi non tutti i numeri sono chiavi possibili. Semplificando molto, per avere la stessa robustezza dell'algoritmo (quindi la stessa difficoltà dal punto di vista computazionale) di un algoritmo simmetrico a 128 bit, l'algoritmo RSA necessita di una chiave dell'ordine di 1024 bit. Gli algoritmi asimmetrici odierni utilizzano in genere chiavi a 1024, 2048 e 4096 bit.

Funzioni hash

Le funzioni hash calcolano un breve riassunto di lunghezza fissa di un testo di lunghezza arbitraria; questo riassunto è chiamato *digest*. Una funzione hash deve essere semplice e veloce da calcolare.

Proprietà di sicurezza

Le funzioni hash sono “funzioni unidirezionali": è molto complesso e lento recuperare un *possibile* testo dato un digest, praticamente impossibile che questo testo sia anche utile e non una sequenza casuale di byte. Questa è una conseguenza del fatto che il testo originale è più grande del digest, quindi infinite sequenze di byte possono dare un certo digest. La modifica di un singolo bit di un messaggio, fa sì che il digest prodotto sia tipicamente completamente diverso rispetto al messaggio originale.

Verifica di integrità

Dato un messaggio M, il suo digest D viene calcolato per mezzo di una funzione hash H: D = H(M)

Una trasmissione sicura del digest D permette di garantire che il messaggio M sia corretto: il destinatario del messaggio M calcola il digest su di esso e lo confronta con quello (D) ricevuto; se sono identici, il messaggio non è stato modificato. È essenziale che il digest venga trasmesso in modo sicuro, altrimenti non c'è alcuna protezione.

Questo meccanismo viene utilizzato anche come verifica di un software scaricato al posto di altri algoritmi meno potenti (ma più veloci) noti come CRC (Cyclic Redundancy Checks), comuni nei protocolli di rete.

Algoritmi hash rilevanti

MD5 (Message Digest #5, 1991)

MD5 richiede una bassa potenza di calcolo (rispetto alla maggior parte degli altri algoritmi hash) e produce un hash a 128 bit. Nel tempo si è progressivamente indebolito fino ad essere del tutto inaffidabile per motivi di sicurezza (attualmente è possibile trovare due messaggi distinti con lo stesso hash MD5 – che viene chiamata *collisione* – in pochi secondi), comunque è comunque utile in applicazioni non crittografiche, ad esempio per scoprire errori di trasmissione al posto del più comune (e debole, ma più veloce) CRC-32.

SHA-3 (Secure Hash Algorithm #3, 2015)

SHA identifica in realtà una famiglia di algoritmi hash sicuri, SHA-3 è l'ultima versione e è il risultato di una competizione tra crittografi. L'algoritmo può essere adeguato agli scopi bilanciando sicurezza e velocità; produce valori hash di 224, 256, 334 e 512 bit. È 2-3 volte più lento di MD5 ma finora non è stato riscontrato alcun punto debole.

Firme (Firme digitali)

Le firme digitali, chiamate firma elettronica nella legislazione di alcuni paesi, calcolano l'hash di un messaggio e lo cifrano (solo l'hash) con la chiave segreta di un algoritmo asimmetrico. Questo hash firmato con la chiave segreta è chiamato *firma*.

Una firma digitale quindi non cripta il messaggio originale, ma permette solo di garantire che il testo corrispondente non sia stato modificato, perché altrimenti il digest cambierebbe.

La firma garantisce inoltre che l'origine possa essere verificata (*non ripudio*), in quanto è cifrato con con la chiave privata di un algoritmo asimmetrico. Ciò significa che chiunque può verificare che il testo sia integro e sia stato effettivamente generato da un determinato soggetto.

Simili alle firme digitali sono i **Message Authentication Codes** (Codici di autenticazione dei messaggi - MAC) che garantiscono l'integrità e l'autenticità di un messaggio. Il digest viene calcolato effettuando l'hashing del testo da proteggere insieme con una chiave di un algoritmo simmetrico

(non vi è alcuna garanzia di non ripudio, quindi è adatto solo quando questa caratteristica non è necessaria). Uno degli algoritmi più conosciuti è HMAC: Keyed-hash MAC.

Infrastrutture a chiave pubblica

I requisiti di riservatezza, integrità e autenticità delle operazioni di crittografia asimmetrica e firma digitale richiedono l'instaurazione di un rapporto di fiducia tra le parti coinvolte. Poiché spesso le parti non hanno avuto contatti prima, una terza parte di (loro) fiducia deve fornire le necessarie garanzie, in altre parole questa terza parte deve assicurare l'autenticità del proprietario di una chiave pubblica.

Una *Certificate Autority* (Autorità di certificatione - CA) fa parte di un modello di fiducia gerarchico che garantisce l'identità dell'utente. Deve essere riconosciuta come parte "fidata" da tutti i soggetti coinvolti nella comunicazione.

Al fine di garantire l'autenticità di un soggetto, la CA firma un *certificato digitale* che lega il soggetto alla sua chiave pubblica (è concettualmente simile a un documento di identità di una persona). La firma applicata dalla CA è una firma digitale, quindi viene calcolata utilizzando la chiave privata della CA. Naturalmente, il certificato digitale può essere convalidato utilizzando la chiave pubblica della CA.

Una *Public Key Infrastructure* (*Infrastruttura a chiave pubblica* - PKI) è una struttura gerarchica di CA, dove ogni CA ha la sua chiave pubblica firmata dalla CA di livello superiore (che utilizza a tale scopo la sua chiave privata), il processo sale fino alla root CA che firma il certificato contenente la sua chiave pubblica utilizzando la sua chiave stessa private (root certificate).

Un modello gerarchico consente di costruire gerarchie di qualsiasi profondità. Ad esempio, all'interno di un Paese si possono avere le autorità di certificazione comunali, …, regionali, fino a una root CA nazionale.

Il certificato della root CA deve essere reso pubblico in più posti e vari modi in modo da rendere impossibile sostituirlo con uno contraffatto. Ad esempio, i sistemi operativi stessi vengono forniti con una serie di certificate di root CA e CA intermedie sicure. La figura 1 mostra un estratto dell'elenco dei root certificate da un sistema operativo MS Windows 10.

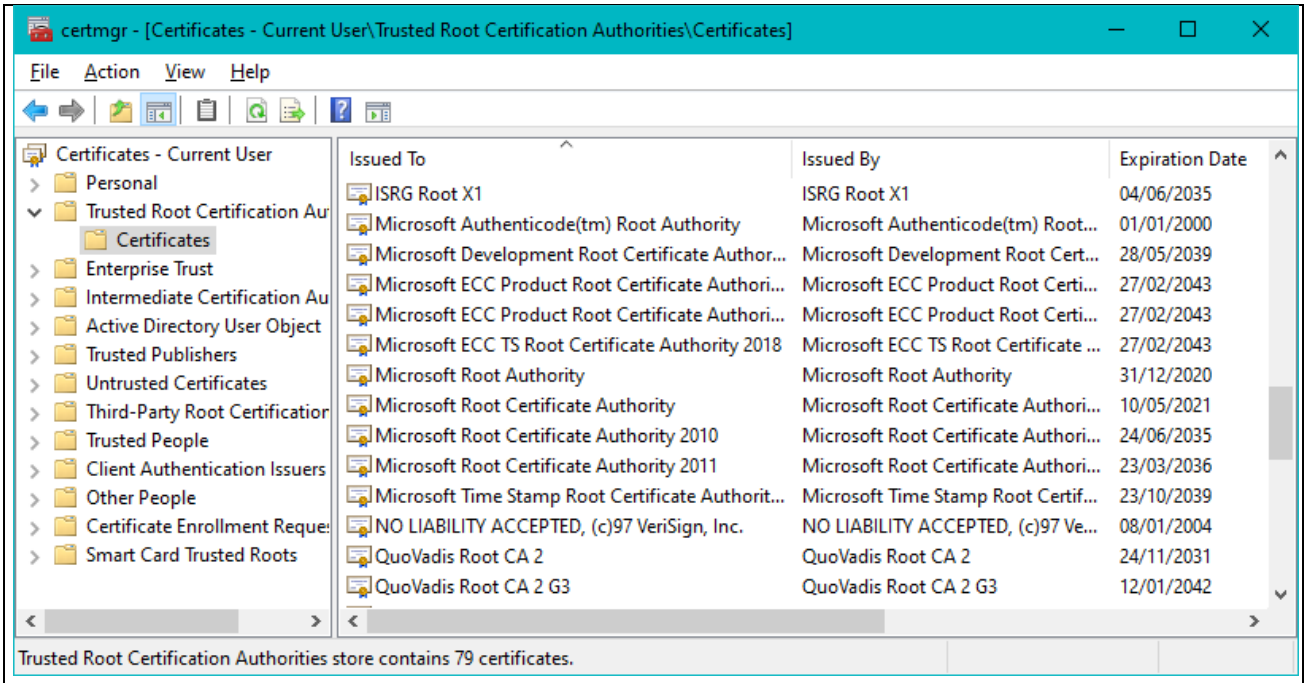

Figure 1. Windows 10 Certificate Manager

Componenti di una PKI

L'entità principale è l'**Autorità di certificazione (CA)**, questa gestisce le chiavi crittografiche e i certificati della PKI. È il componente che detiene le chiavi reali e che quindi esegue le operazioni crittografiche. Ovviamente, in questo senso è anche l'elemento più critico dell'autorità di certificazione.
Il **Certificate server** è la directory (archivio) in cui i certificati utente sono pubblicati e resi ricercabili. La directory segue lo standard X.500 e viene interrogata utilizzando il protocollo LDAP.

La **Registration Authority** (**Autorità di registrazione - RA)** gestisce le richieste di certificati e fornisce i certificati emessi. Si occupa della distribuzione di token crittografici e software (è il front-end della CA). È come uno sportello, ad esempio, al quale un cittadino può recarsi per richiedere e ottenere un certificato digitale. Può essere un servizio fisico o un sito web, la tipologia è associata a diversi livelli di sicurezza. In aggiunta, una RA generalmente può anche assistere l'utente nella generazione della coppia di chiavi e nella gestione sicura di questi certificati, fornendo software o strumenti hardware che supportano il richiedente nella generazione di firme, cifratura di documenti, verifica delle firme, decifratura delle informazioni.

Il certificato digitale

La struttura e il formato del certificato digitale sono descritti dallo standard ITU-T X.509 (è lo standard numero X.509 dell'International Telecommunication Union).
Un certificato digitale X.509 deve contenere almeno i seguenti campi:

DN (distinguished name) utente: Caio Giulio Cesare
CN (common name): Giulio Cesare
O (organization): Governo
UO (organization unity): Senato
C (country): Impero Romano

Questo è il modello classico di un certificato che soddisfa lo standard X.500, di cui l'X.509 è una parte. Un certificato può avere altri attributi (es. Email) e potrebbe esserci un indirizzo IP o un URL che corrisponde a un percorso di directory X.500 (es. LDAP – Lightweight Directory Access Protocol): IT / Impero Romano / Governo / Giulio Cesare / Caio Giulio Cesare /

Un certificato digitale X.509 deve inoltre contenere almeno i seguenti campi:

DN Emittente: nome dell'Autorità in formato X.500 (l'Autorità di certificazione ha anche un proprio nome distinto che compare nel certificato. Ovviamente il formato con cui è rappresentato è lo stesso del DN del soggetto a cui si applica il certificato .

Numero di serie: numero seriale del certificato posseduto dall'Autorità (ogni certificato è identificato univocamente dal numero seriale assegnatogli dall'Autorità. La coppia (*numero di serie*/*DN dell'emittente*) identifica in modo univoco un certificato tra tutti quelli che possono essere emessi in tutto il mondo.

NotBefore: data di attivazione

NotAfter: data di scadenza

Si tratta di due informazioni strettamente correlate tra loro. Un certificato ha quindi una data di inizio vita e una data di scadenza, oltre la quale non è più valido. Il motivo deriva dalla volontà di proteggere le chiavi da un uso eccessivo.

Public Key: la chiave pubblica del soggetto certificato (il certificato contiene i dati anagrafici e la chiave pubblica)

Public Key Algorithm: l'algoritmo per cui è adatta la chiave pubblica

Signature Algorithm: l'algoritmo utilizzato dall'Autorità per la firma del Certificato

Signature: la firma apposta sul Certificato dall'Autorità di Certificazione (quindi l'elemento che garantisce l'autenticità del certificato stesso)

KeyUsage: gli scopi per cui può essere utilizzata la chiave

Generazione e Revoca di Certificati

Lo standard RFC 2314 (da PKCS # 10) definisce il formato del messaggio (file) *certificateRequest* (richiesta di certificato) e contiene la chiave pubblica e i dati del richiedente (entrambi forniti a una RA). Il messaggio di richiesta richiede la cosiddetta *prova di possesso* della chiave privata: il *certificateRequest* deve essere firmato con la chiave privata del richiedente. L'autorità, verificando tale firma, si verifica sostanzialmente che il richiedente sia effettivamente in possesso di tale chiave (altrimenti il richiedente non avrebbe potuto produrre tale firma).

Alla data corrispondente al "notAfter", il Certificato è considerato "scaduto". La scadenza è un meccanismo che serve anche a garantire che le stesse chiavi asimmetriche associate al certificato digitale non possano essere utilizzate per un periodo di tempo eccessivo, che potrebbe esporle ad una serie di attacchi di crittoanalisi.

La scadenza può riferirsi a:

il certificato (l'associazione oggetto-chiave)

le chiavi associate, in relazione all'uso a cui sono destinati (una chiave può essere utilizzata solo per determinati scopi e non per altri)

Un certificato scaduto non può essere utilizzato dal suo titolare, poiché è scaduto il suo utilizzo è in pratica vietato. In realtà nulla vieta al titolare del certificato di utilizzarlo per altri scopi ma tutti coloro che comunicano con lui rileveranno in qualche modo che tale certificato è scaduto e non lo accetteranno. Il certificato è ovviamente disponibile per verificare e decifrare i messaggi emessi prima della scadenza. In alcuni casi è possibile "rinnovare" un certificato, il che di solito non è automatico.

A volte un certificato può essere reso non valido prima della scadenza tramite *revoca* (questo accade prima del valore di "notAfter"). Il motivo potrebbe essere, ad esempio, che la chiave associata al certificato è stata compromessa, il soggetto non è più associato all'azienda o ha cambiato unità, la CA che ha emesso il certificato (o una delle sue CA di livello superiore) è stata compromessa o altri motivi. Un certificato revocato non può più essere utilizzato, tranne quando viene semplicemente tenuto "in attesa" (è detto "on hold"). Lo stato "on hold" è una particolare forma di revoca in cui il certificato viene inserito temporaneamente nella Certificate Revocation List (CRL) per motivi specifici (quali "possibile compromissione della chiave", "richiesta di revoca da parte di soggetto diverso dal titolare", "non utilizzo temporaneo del certificato " es. per ferie)", ecc. e la sua validità potrebbe essere ripristinata in seguito. In caso di "riattivazione" dalla sospensione viene inserita una nuova voce nella CRL con un apposito codice di motivazione: *removeFromCRL*, poiché nessuna voce può mai essere eliminata dal CRL.

Per informare gli utenti della PKI che un certificato ha perso validità, il suo numero di serie è elencato nella CRL. Tale elenco viene rilasciato periodicamente dalla CA emittente e propagato (attivamente o passivamente) a tutti gli utenti, in modo che tutti i soggetti in possesso di tale certificato non possano più utilizzarlo. Il CRL è conforme allo standard ITU-T X.509 - RFC3280 - RFC4325, il che significa che il suo formato è standardizzato. Viene sottoscritto periodicamente dalla CA e comprende un campo denominato "nextUpdate" (prossimo aggiornamento) che contiene la data e l'ora in cui sarà disponibile la nuova CRL.

Se la data di nextUpdate appartiene al passato significa che il nostro CRL è "scaduto", se la data appartiene al futuro sappiamo quando dovremo aggiornare il nostro elenco di certificati revocati. Ogni certificato revocato include il motivo della revoca. Il CRL viene fornito come servizio tramite il protocollo LDAP.

Poiché le CRL possono diventare piuttosto grandi, soprattutto se l'autorità di certificazione ha centinaia di migliaia o addirittura milioni di utenti, queste possono essere partizionate e la CA fornisce solo parti dell'elenco di revoche di certificati, ad esempio facendo riferimento a un determinato periodo di tempo. Pertanto, quando è necessario verificare un certificato, il cittadino scarica l'unico pezzo di CRL di interesse. Invece di scaricare un CRL o una parte di esso, un servizio che implementa l'*Online Certificate Status Protocol* (OCSP - RFC 2560) può essere utilizzato per determinare se un determinato numero di certificato è presente nell'elenco CRL. Le risposte sono firmate dalla CA per essere attendibili.

La catena della fiducia

Per verificare un certificato, un soggetto deve verificare tutta la catena delle CA intermedie a partire da una attendibile, possibilmente la root CA. Prima il certificato da controllare viene verificato rispetto alla CA emittente, quindi il certificato stesso di questa CA viene verificato rispetto alla CA precedente nella gerarchia e così via fino a una CA intermedia attendibile (dall'utente) o alla root CA stessa.

KeyUsage

In realtà, le chiavi possono essere utilizzate per diversi scopi e un certificato specifica gli usi consentiti, tra questi abbiamo:

nonRepudiation: la chiave può essere utilizzata per produrre una firma con valore legale che può poi essere opponibile a terzi come firma autografa

dataEncipherment: la chiave può essere utilizzata per cifrare i dati

keyAgreement: la chiave può essere utilizzata per scambiare le chiavi in modo sicuro

TimeStamping

Una PKI può anche essere utilizzata per contrassegnare con una marca temporale timestamp) un documento secondo il *Time Stamp Protocol* (RFC 3161), per dimostrarne l'esistenza in un momento nel passato. Una CA firmerà il documento e includerà il timestamp. La firma garantisce inoltre che il documento non sia stato alterato dopo l'apposizione del timestamp. Un timestamp può estendere la validità di una firma elettronica oltre la durata del certificato digitale associato alle chiavi utilizzate, ad esempio richiedendo l'apposizione di un nuovo timestamp sul contratto ogni anno che passa. L'apposizione sarà effettuata annualmente utilizzando tecniche crittografiche con forza adeguata al momento in cui viene applicata la firma stessa.

Meccanismi e formati standard

PKCS #7 - Public key cryptography standard #7

PKCS #7 descrive il formato dei messaggi/file firmati, crittografati (con crittografia simmetrica e asimmetrica) e verificabili tramite MAC (digest)

S/MIME - Secure Multipurpose Mail Extension

È un'estensione dello standard MIME (RFC 2045, 2046 e 2047 come estensioni di RFC 822) e descrive la crittografia dei messaggi di posta. Le entità PKCS #7 sono sostanzialmente tradotte in nuove entità MIME e quindi descrivono nuovi contenitori in cui possiamo trovare porzioni di messaggi di posta elettronica firmati o crittografati o allegati firmati o crittografati.

Formati di scambio e archiviazione

PKCS #12 (Personal Information Exchange Syntax Standard #12 - Standard della sintassi per lo scambio di informazioni personali n. 12) definisce un formato per lo scambio e l'archiviazione sicura (su file) di chiavi crittografiche e certificati digitali (catena di certificati attendibili). L'archiviazione è sicura perché si basa sulla crittografia sia delle chiavi sia dei certificati nonché dell'intera catena di certificati di cui un soggetto sceglie di fidarsi. È il formato utilizzato da browser, client di posta elettronica e sistemi operativi quando un utente esporta una firma e una chiave di crittografia e un certificato, ad esempio per una copia di backup o per copiarlo su un altro dispositivo.

Un messaggio PKCS #12 cifra i dati con un algoritmo simmetrico la cui chiave deriva da una password fornita dall'utente. Garantisce l'integrità del contenuto tramite HMAC o tramite una firma digitale.

I vantaggi dell'utilizzo di PKCS #12 sono:

nessun costo di implementazione in quanto è un formato disponibile in molte librerie per molti dispositivi (dai computer ai cellulari)

è già supportato dalla maggior parte dei sistemi operative e software, questo significa che è immediatamente disponibile

utilizza algoritmi simmetrici "forti".

è portabile su diversi dispositivi, è possibile spostare un file su altri dispositivi o copiarlo su tutti i dispositivi che utilizziamo per avere le chiavi sempre disponibili

Ci sono anche alcuni svantaggi:

può essere cancellato, copiato, trasmesso, è un problema importante perché può essere rubato, cancellato, copiato per sottrarlo o semplicemente reso non disponibile

la chiave privata deve essere utilizzata direttamente dal software di firma/decrittografia, il software usa decripta questo file e accede direttamente alla chiave, quindi c'è un momento in cui la chiave è in chiaro sul sistema, non protetta, anche se forse è solo la memoria volatile nell'area meno accessibile; tuttavia è in qualche modo disponibile in chiaro sul sistema e questo è una limitazione importante di questo formato

è possibile eseguire attacchi di brute force, ad esempio, nel caso in cui qualcuno rubi il file PKCS #12, può provare tutte le password possibili in un altro sistema informatico

Token crittografici

I token crittografici sono dispositivi hardware che implementano nativamente algoritmi di crittografia. Non consentono l'accesso diretto al materiale crittografico dall'esterno, sono gli unici abilitati ad accedere alla chiave e ad utilizzarla. Inoltre queste operazioni comandi possono essere generalmente eseguiti solo dopo che l'utente, e il sistema per conto dell'utente, si è autenticato sul token, ad esempio inserendo un PIN. I token possono avere un piccolo sistema operativo attraverso il quale l'utente richiede operazioni di generazione di chiavi, crittografia e firma dei dati.

La distruzione sicura del token spesso richiede non solo la cancellazione, ma anche l'impossibilità di accedere alle aree di memoria dove è stata scritta la chiave, così da evitare qualsiasi forma di attacco che oggi non si possono prevedere. Il sistema operativo e l'hardware dei token possono essere certificati secondo specifici standard di sicurezza.

Il meccanismo challenge-response

Un meccanismo di challenge/response (sfida/risposta) consente a due parti di autenticarsi senza esporre informazioni utili sul canale di comunicazione. L'idea è che una parte A "sfida" l'altra parte B ad operare su un numero casuale (chiamato *nonce* perché è un numero usato solo una volta) con una propria chiave, ad esempio mediante una funzione unidirezionale H. Allora B restituisce il risultato di N=H(*nonce*) ad A. Se B restituisce ciò che A si aspetta, A è sicuro dell'identità di B. Nel caso più semplice, B cripta il nonce con una chiave segreta nota ad A. A, che ha la stessa chiave, a sua volta applica la funzione al nonce utilizzando la stessa chiave e verifica che il numero che gli è stato assegnato da B corrisponda quello che ha calcolato.

Si tratta di un'autenticazione basata su una password che non è stata inviata sul canale, quindi non è possibile risalire alla chiave. Ciò che rende efficace il meccanismo è il fatto che il numero casuale è diverso per ogni sessione. Poiché le informazioni lette sul canale sono diverse per ogni sessione, non sono utili per l'autenticazione successiva. Un vantaggio è che non è necessaria alcuna sincronizzazione (a differenza del meccanismo One-Time-Password, dove è importante conoscere il punto di partenza del Pad da utilizzare). Poiché non è facile da usare da parte degli esseri umani, viene in genere utilizzato nell'autenticazione da macchina a macchina. Questo meccanismosi trova nei meccanismi di autenticazione di rete dei router ADSL al server di autenticazione del provider.

Domande:

In file separato.

Attachments:

Presentazione Powerpoint in file separato

Referenze:

Ogni argomento ha una sua pagina su Wikipedia, a volte già troppo approfondita per gli scopi di questo corso; i libri sono troppo specializzati e sono utili solo dopo una formazione molto approfondita sull'argomento specifico.