Crittografia

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

Introduzione

La crittografia è la scienza di *criptare* (o *codificare*) un messaggio binario m (i.e. un testo in chiaro e comprensibile) in un *critto-testo* o *cypher-text* c (un testo incomprensibile), il quale può essere *decriptato* (o *decodificato*) per ottenere nuovamente m.
In generale, la crittografia è un arte molto antica. Esistono, infatti, tracce risalenti all'antica Grecia.

Crittosistema

Un crittosistema è una coppia di algoritmi, uno per cifrare ed uno per decifrare. Questi due algoritmi hanno una caratteristica in comune: i parametri in input, ovvero il **testo** (i.e. una stringa di bit, cifrata nel caso di decifratura) e la **chiave** (i.e una stringa di bit) e restituiscono un solo valore in *output*. In particolare, notiamo:

- Un algoritmo $\mathcal{E}$ per la **codifica** del messaggio, per cui data una chiave k e un testo m, questo viene codificato attraverso $\mathcal{E}(m, k)$, il quale produce m_k, ossia il *messaggio criptato*.
- Un algoritmo $\mathcal{D}$ per la **decodifica** del messaggio, il quale data una chiave k^1 e un crittotesto m_k, questo viene decodificato attraverso $\mathcal{D}(m_k, k^1)$, che produce m (i.e. il messaggio comprensibile) se **precise condizioni** legano k con k^1.

Queste precise condizioni ci permettono di distinguere i due tipi di crittografia: la **crittografia simmetrica** e la **crittografia asimmetrica**.

> Il termine chiave è associato storicamente ai chiavistelli, quindi l'oggetto che permette di aprire. Noi intenderemo quest'ultima secondo la sua versione digitale, ossia una stringa di bit.

> Possiamo definire la crittoanalisi come la scienza (i.e tecnica) finalizzata a rompere un crittotesto pur senza conoscere la chiave.

Segretezza di una chiave

Abbiamo capito che l'algoritmo (in particolare, crittosistemi) sarà **pubblico** (la specifiche degli algoritmi $\mathcal{E}$ e $\mathcal{D}$) e manterremo segreta la chiave se vogliamo mantenere segreto il contenuto di un crittotesto.
Tuttavia, sappiamo anche che vi è sempre la possibilità (se pur minima) di indovinare la chiave attraverso un approccio di bruteforce, per cui maggiore è la lunghezza k del segreto, minore è la probabilità di scoprire quest'ultimo. In particolare, data una chiave di lunghezza k, esiste probabilità di indovinare il segreto uguale a:

$$Pr[Guess(k)] = \frac{1}{2^k}$$

Crittografia simmetrica

La prima tipologia di crittografia che studiamo è la storica **crittografia simmetrica**, la quale prevede che la precisa condizione di cui scrivevamo precedentemente sia uguale alla relazione di uguaglianza tra k e k^1.
Formalmente, possiamo definire un crittosistema simmetrico, se l'unico modo per ottenere il testo in chiaro m da un *crittotesto* si basa sull'utilizzo della **stessa chiave** $k^1 = k$ utilizzata per costruire il *crittotesto*.
Conseguentemente, per ogni $k \neq k^1$ segue un messaggio diverso da m.
Riassumendo, le **caratteristiche** della crittografia simmetrica sono le seguenti: (i) una chiave $k = k^1$ (tipicamente con una lunghezza uguale a 128 o 256 bit) e (ii) la sua velocità.
Alcuni crittosistemi simmetrici sono, per esempio, il *Cifrario di Cesare*, *DES* e *3DES*.

Un **assunzione** fondalmentale si basa sull'assumere che ogni crittosistema sia *pubblico* mentre le chiavi crittografiche sono tipicamente mantenute segrete. Ovviamente mantenere un chiave segreta permette comunque ad un attaccante di provare ad "*indovinare*" la chiave secondo un approccio *bruteforce*. Un approccio di questo tipo ha probabilità di riuscita pari a $\frac{1}{2^n}$, un numero che sicuramente disincentiva rispetto l'attacco ma che ci suggerisce che un modo per irrobustire una chiave sia aumentare la sua lunghezza.

Crittografia simmetrica in rete

Supponiamo che $Alice$ e Bob vogliano comunicare in modo segreto e che chiunque abbia una propria chiave segreta, la quale non deve essere rivelata nello stesso modo in cui non viene rivelata una propria password (dalla quale differisce esclusivamente e banalmente per la lunghezza, la quale è *più grande* per la chiave).
Supponiamo che sia $Alice$ che Bob abbiano una chiave, rispettivamente k_a e k_b, utili a cifrare i messaggi da inviare. In questo modo, un attaccante $Charlie$ può al più intercettare il messaggio criptato, garantendo la proprietà di segretezza tra i due intelocutori.
Purtroppo, il protocollo così come indicato *non è funzionante* poiché Bob, per leggere il messaggio criptato da $Alice$, necessita della conoscenza di k_a, chiave non al momento conosciuta da Bob.

Come potrebbe Bob conoscere la chiave di $Alice$, ossia k_a? $Alice$ potrebbe inviare preventivamente la sua chiave a Bob, tuttavia ciò richiede un notevole grado di fiducia; d'altra parte proteggere la chiave k_a con un altra chiave riproporrebbe il problema.
Questo è il vero limite della crittografia simmetrica, il quale si basa sulla chiave stessa, che deve essere conosciuta sia da chi cifra il messaggio che da chi decifra il messaggio stesso.

Limiti della crittografia simmetrica

Come accennato precedentemente, il **limite fondamentale** della crittografia simmetrica si basa sulla condivisione di un segreto iniziale tra $Alice$ e Bob. Tipicamente il segreto condiviso per la comunicazione segreta non è la chiave a *lungo-termine* di uno degli interlocutori, bensì una chiave a *breve-termine* relativa ad entrambi, ossia k_{ab}. In definitiva, Alice utilizza la sua chiave a *lungo-termine* esclusivamente per negoziare un segreto condiviso con l'interlocutore, ovvero una chiave a breve termine per la singola sessione, ossia k_{ab}.

> [$Esame$] Qual è il limite fondamentale della crittografia simmetrica?

> [$Esame$] Si formalizzi una definizione di crittografia simmetrica.

Crittografia asimmetrica

D'altra parte, studiamo adesso la seconda e più recente tipologia di crittografia: la **crittografia asimmetrica**.
All'interno di quest'ultima crittografia non esiste una sola chiave bensì una coppia di chiavi in cui l'una è l'inversa dell'altra. Le due chiavi a cui facciamo riferimento sono k e la sua inversa k^{-1} (non riferito

all'*operazione matematica* ma così denotata).
Si noti che ciascuna chiave **non si può ricavare dall'altra** (il che è un *problema intrattabile*), motivo per cui la coppia di chiave va generata insieme (i.e. *monoliticamente*).
Attraverso un crittosistema asimmetrico, l'unico modo per ottenere il testo in chiaro da un *crittotesto* si basa sull'utilizzo dell'inversa della chiave utilizzata per costruire il *crittotesto*. Conseguentemente, per ogni $k^1 \neq k^{-1}$ segue un messaggio diverso da m.

Alcuni crittosistemi asimmetrici sono, per **esempio**, *DSA* ed *RSA*.
Caratteristiche della crittografia asimmetrica sono (i) chiavi tipicamente di lunghezza pari a $1024\ bit$ e (ii) una maggior lentezza rispetto la crittografia simmetrica.

Crittografia asimmetrica in rete

Attraverso il paragrafo precedente abbiamo capito che esiste un limite nel momento in cui $Alice$ e Bob vogliano comunicare in segreto secondo la crittografia simmetrica, la quale li costringe a negoziare un segreto a *breve-termine* facendo uso delle loro chiavi a *lungo-termine*.
D'altra parte, secondo la crittografia asimmetrica notiamo un miglioramento della situazione. Attraverso la crittografia asimmetrica, ipotizziamo che $Alice$ sia munita della sua coppia di chiavi (k_a, k_a^{-1}), rispettivamente la coppia *chiave-pubblica* (i.e. conosciuta da *tutti*), *chiave-privata* (i.e. conosciuta dalla sola $Alice$ e, *per assunto*, da Bob) e viceversa per Bob.
Ipotizzando, quindi, la medesima conversazione segreta fra $Alice$ e Bob, quale chiave dovrebbe utilizzare $Alice$ per inviare un messaggio a quest'ultimo? Notiamo, innanzitutto, che se $Alice$ utilizzasse la sua chiave pubblica k_a per criptare il messaggio, allora solo egli stessa potrebbe leggerne il contenuto attraverso la sua chiave privata k_a^{-1}. D'altra parte, se $Alice$ utilizzasse la sua chiave privata k_a^{-1} per criptare il messaggio, allora chiunque potrebbe leggerne il contenuto attraverso sua chiave pubblica k_a.
Soluzione è quindi far sì che $Alice$ utilizzi la chiave pubblica del destinatario stesso (Bob), k_b. In questo, Bob sarà l'unico a poter decriptare il messaggio secondo la sua chiave privata k_b^{-1}, quindi leggere quest'ultimo.

Limiti della crittografia asimmetrica

Il limite della crittografia asimmetrica si basa sulla *verifica* che la chiave pubblica del destinatario appartenga effettivamente ad esso.
Il protocollo, infatti, funziona a partire dall'ottenimento dell'informazione relativa alla chiave pubblica del destinatario Bob, informazione che potrebbe non essere corretta (**limite**). Immaginiamo, quindi, che un attaccante $Charlie$ con la sua coppia di chiavi (k_c, k_c^{-1}) sia riuscito a fornire ad $Alice$ l'informazione k_c, facendo credere a quest'ultima che sia la chiave pubblica di Bob, k_b. In tal caso, $Alice$ dopo aver criptato il messaggio attraverso k_c invierebbe il tutto; il messaggio criptato $\{m\}_{k_b}$ verrebbe quindi intercettato da $Charlie$, il quale può decriptare il messaggio e leggere il contenuto destinato in origine a Bob.
Supponendo che anche Bob riceva $\{m\}_{k_b}$, allora decriptando il messaggio attraverso la sua chiave privata k_b^{-1}, egli leggerà un contenuto incomprensibile.

Il problema rispetto l'ottenimento della chiave pubblica di Bob per $Alice$ viene risolta attraverso la **certificazione**.

> A partire dalle nostre più rosee ipotesi, possiamo affermare che esiste la possibilità di decriptare e leggere il messaggio esclusivamente se in possesso della chiave privata di Bob. Per noi, infatti, la cifratura è uno strumento che assumiamo funzioni sempre.

Crittosistema sicuro

Possiamo definire un crittosistema sicuro quando, per ogni messaggio m criptato attraverso una qualsiasi chiave k ($\mathcal{E}(m,k)$), decriptando quest'ultimo secondo una qualsiasi chiave k_1 diversa da k (*caso simmetrico*) o k^{-1} (*caso asimmetrico*) ($\mathcal{D}(\mathcal{E}(m,k),k_1)$), allora ottenere n non aumenta le probabilità di ottenere il

messaggio m o porzioni di esso. In altre parole, un crittosistema è sicuro quando esiste una sola chiave k (k per il *caso simmetrico* o k^{-1} per il *caso asimmetrico*) in grado di aprire il crittotesto.

Hash crittografico

Teoricamente, possiamo definire un hash crittografico come una funzione in grado di soddisfare i seguenti requisiti:

- Il calcolo dell'hash per un messaggio m è operazione veloce;
- Il calcolo del messaggio m a partire dal suo hash $hash(m)$ è un problema intrattabile;
- Cambiare un bit all'interno di m, ottenendo m_1, fa sì che l'hash calcolato per m_1 sia completamente diverso rispetto l'hash di m;
- Non è possibile trovare lo stesso $hash(m_1) = hash(m_2)$, per due messaggi m_1, m_2 tale che $m_1 \neq m_2$.

Com'è possibile implementare un funzione hash in grado di soddisfare questi requisiti? E' importante non confondere la defizione teorica della funzione hash con una sua ipotetica implementazione, in ogni caso le implementazioni di funzioni hash nel tempo sono state diverse.
Esempi di funzioni hash sono:

- **SHA-1**, la quale ha dimostrato nei limiti nel tempo, ovvero la *non* capacità di soddisfare i requisiti descritti nella teoria;
- **MD5**, che allo stesso modo ha dimostrato nei limiti nel tempo (sono state trovate delle *collisioni*).

Un esempio di utilizzo dell'hash: CTSS + Hashing

Il primo sistema di autenticazione (orientato alla *multi-utenza*) è **CTSS**, sviluppato dall'**MIT** alla fine degli anni $'60$. Si basava su una tabella in cui fossero definiti username e password, attraverso cui l'utente che voleva accedere al sistema avrebbe dovuto inserire la sua password, una stringa che doveva coincidere quella definita all'interno della tabella stessa.
Questo è un problema nella misura in cui un attaccante può essere in grado di arrivare alla tabella, motivo per cui, 7 anni dopo (a Cambridge) nasce l'idea di potenziare il sistema secondo una funzione hash.

> E' importante non confondere l'hash con la crittografia, le quali si differenziano l'una dall'altra. Una differenza importante si basa sul fatto che un messaggio criptato può essere decriptato attraverso la corretta chiave, diversamente da un hash crittografico per cui non esiste la possibilità (concreta) di ritornare al messaggio di partenza.

Salting

Come sappiamo, minore è la lunghezza della password e minori saranno i tentativi da svolgere prima di indovinare, anche se alla generazione di una password segue la generazione dell'hash e la successiva comparazione. Dunque, il fatto che sia presente l'hash non rende impensabile un tentativo di *bruteforce*.
Per questo motivo, nasce il concetto di **salting** (*sale*) nasce per incrementare la lunghezza di una password, per *condire*. In questo modo è possibile rendere una password più robusta rispetto *attacchi a dizionario*.

> Storicamente il sale era composto da $12\ bit$, oggi insufficienti.

Aggiunta di una password in Unix

E' interessante notare l'utilizzo del sale all'interno del mondo Unix e, più in generale, come avvenga aggiunta una password all'interno di un sistema come quest'ultimo. Ipotizzando che un *sys admin* generi un nuovo utente con una nuova password, allora sappiamo essere necessario il calcolo della sua versione *hash*, la quale verrà memorizzata nel file *etc/passwd*.
Ci soffermiano, quindi, su come Unix, storicamente, proponeva di svolgere l'hash di quest'ultima password. Sostanzialmente il tutto si basa su una certa tipologia di funzione di encryption, una funzione di *One-Way Encryption*, ovvero `crypt(3)`.
Quest'ultimo è in grado di fornire una versione "*hash*" (formalmente non un *hash*) della password, ovvero una versione di quest'ultima per cui è intrattabile eseguire il *decrypt*. I vantaggi di utilizzare una funzione crittografica e non un hash sono presto detti: attraverso `crypt()`, infatti, possiamo godere della prime e seconda proprietà sopra elencate relative all'hash crittografico e dimenticarci delle altre proprietà di difficile implementazione.
In generale, `crypt(3)` necessita di una stringa da crittografare ed una chiave secondo cui svolgere l'*encrypt*. Questi due parametri sono rispettivamente:

- Una stringa formata da sale ed una stringa costante (solitamente formata soli zero); in particolare il sale era formato da due caratteri scelti randomicamente dal set (a-zA-Z0-9) used to perturb the algorithm in one of 4096 different ways.
- Una chiave di $56\ bit$ ricavata dai $7\ bit$ meno significativi dei primi 8 caratteri della password.

User ID	Salt	Ecrypted Password(Salt + 0s, Key)
jax	7a	r8f94hf7fdvfd8d

Verifica di una password in Unix

La verifica di una password all'interno di un sistema Unix si basa sulla seguente procedura. Dopo che l'utente ha inserito user id e password, il sistema cerca lo user id all'interno della tabella in cui memorizza quest'ultimi dati, trovando conseguentemente la tupla contenente: userid, il sale e la ecrypted password. A questo punto, viene effettuata il confronto tra la live-password, generata a partire dal precedente sale e la password digitata, e la encrypted password.

Freshness

Un ulteriore pezzo di puzzle relativo alla crittografia è dato dalla **freshness** (i.e. *freschezza*, *essere recenti*), non una proprietà ma un **attributo** di un altra proprietà, in particolare dell'autenticazione e la segretezza). Essa, quindi, indica la "freschezza" per la proprietà (e.g. dell'autenticazione).

Il legame tra freshness e **confidenzialità**, ossia il legame tra la freshness ed un segreto è molto importante poiché essendo possibile effettuare il *bruteforce* per ogni segreto, procedura per cui è necessario del tempo, allora maggiore sarà il tempo trascorso dalla generazione del segreto, maggiore il tempo a disposizione per effettuare *bruteforce*. Per questo motivo, un segreto *fresco* è più robusto rispetto ad un attacco bruteforce.
D'altra parte, il legame tra freshness ed **autenticazione** esiste, poichè ciò che autentico ad un istante t potrebbe non essere autenticato ad un istante $t+1$ (i.e. potrebbe non esser più lui). Si pensi, per esempio, se un bancomat mantenesse l'autenticazione di un utente dopo che quest'ultimo ha svolto la determinata operazione e si è allontanato da esso; in quel caso, qualcun'altro potrebbe svolgere operazioni per conto di quest'ultimo, il che non è desiderabile.

Come è possibile **implementare** la freshness? Sostanzialmente esistono due modi per implementare la freshness, ovvero:

- Attraverso un **timestamp**, o marcatore temporale, scrivendo la data di produzione per una chiave. Quest'ultima viene analizzata e poi accettata solo se "*rientra nella sua finestra temporale*". Si noti che

i timestamp hanno senso finchè tutti noi abbiamo un orologio sincronizzato. La sincronizzazione degli orologi su sistemi distribuiti è un problema risolto con un protocollo che è necessario sia sicuro.

- Attraverso **Nonce** (*Number random that is used only Once*), basato sull'idea che si utilizzi per veicolare la freshness sull'autenticazione. Supponiamo che un autenticazione di $Alice$ venga effettuata attraverso l'invio di una chiave segreta; se un attaccante $Charlie$ ha intercettato quest'ultima chiave, allora $Charlie$ potrebbe autenticarsi come Bob. Una **misura** per evitare questo attacco, chiamato **replay attack**, si basa sulla freshness.

- i $Alice \rightarrow (Nonce_A) \rightarrow Bob$ *attraverso una condivisione sicura*

- $i+1$ $Bob \rightarrow ((Nonce_A\, Msg)_K) \rightarrow Alice$

- $i+2$ $Alice$ *autentica* Bob.

- $i+3$ Bob *esce. Inizio del tentativo di replay attack da parte di* $Charlie$.

- $i+4$ *Il messaggio di* Bob *ad* $i+1$ *viene intercettato da* $Charlie$.

- $i+5$ $Charlie \rightarrow ((Nonce_A\, Msg)_K) \rightarrow Alice$

- $i+6$ $Alice$ *non accetta più* $Nonce_A$ *poiché non rientra nella sua finestra temporale*

> Una OTP (*One Time Password*) è un codice numerico generato randomicamente che può utilizzato usato solo una volta. Può usare utilizzato come *freshness* poiché cambia continuamente e si basa su una sincronizzazione preconcordata, motivo per cui ciascuno dei due interlocutori conosce questo codice ed a quale intervallo di tempo si riferisce.