PROVA FINALE – RETI CRITTOGRAFIA

0) NETWORK SECURITY

Una **COMUNICAZIONE SICURA** gode di alcune proprietà:
- **Confidenzialità** → solo l'inviante e il ricevente devono essere in grado di capire i contenuti del messaggio (questo richiede che il messaggio venga criptato, per non essere compreso da chi cerca di intercettarlo);
- **Integrità del messaggio** ("message integrity") → contenuto del messaggio non deve essere alterato nel percorso;
- Autenticazione End-Point → inviante e ricevente devono confermare la loro identità nella comunicazione.

1) CRITTOGRAFIA

La **CRITTOGRAFIA** permette all'inviante di modificare i bit originariamente inviati (ovvero il "PLAINTEXT" o "cleartext") per non farli captare da possibili intruder, facendo capire il messaggio originale solo al ricevente: tramite un ALGORITMO DI CRIPTAZIONE, il plaintext viene criptato in CIPHERTEXT. L'algoritmo di criptazione è però standardizzato in Internet, ovvero ognuno può conoscerlo: per garantire la confidenzialità, si usano le **CHIAVI** (stringhe di caratteri date in input all'algoritmo di criptazione). Ipotizzando lo scambio di messaggi da A a B, l'algoritmo di criptazione prende in input la chiave di A (K_A) e il plaintext che A vuole mandare (m), e dà in output il ciphertext, ovvero $K_A(m)$; quando il ciphertext arriva a B, l'algoritmo di decriptazione con la chiave di B ridà il plaintext, ovvero $\boldsymbol{K_B(K_A(m)) = m}$. Ci sono i sistemi di crittografia a:

A. **CHIAVE SIMMETRICA** → A e B condividono la stessa chiave (identica e segreta); si possono verificare 3 tipi di attacchi da parte dell'intruder:
   - Ciphertext-only = l'intruder ha accesso solo al ciphertext senza sapere nulla del plaintext;
   - Known-plaintext = l'intruder conosce parte del plaintext;
   - Chosen-plaintext = l'intruder sceglie il messaggio di plaintext e ottiene il corrispettivo ciphertext.

   In passato ci sono stati vari algoritmi a chiave simmetrica (es. "Caesar cipher", "Mono-alphabetic cipher" e "Poly-alphabetic cipher"), ma **oggi** si usano "stream ciphers" [di cui non parliamo] e "**BLOCK CIPHERS**" (usati in PGP [email sicure], SSL [connessioni TCP sicure] e IPsec).
   Nel block cipher (o "cifrario a blocchi"), il plaintext è processato in blocchi da k-bit (detti "CHUNK") [es. se $k = 64$, il messaggio è partizionato in chunk da 64-bit e ogni blocco è criptato indipendentemente]. Teoricamente ci dovrebbe essere una tabella con tutte le corrispondenze 1-a-1 tra chunk di plaintext [input] e chunk di ciphertext [output], ma questa diventa dispendiosa con k grandi (es. $k = 64$); quindi nella pratica, i block ciphers usano funzioni che simulano le tabelle randomicamente permutate. Quindi, il plaintext viene partizionato in chunk da k-bit, criptati dalle funzioni-tabella, poi i chunk di ciphertext vengono riassemblati e permutati, e l'output di ciphertext completo viene dato in pasto di nuovo all'algoritmo (nuovo ciclo uguale al precedente, ma con ciphertext in input al posto dei plaintext).

   Nella pratica però si criptano lunghe sequenze, dove 2 o più blocchi di plaintext possono ripetersi (es. la parola HTTP potrebbe comparire più volte) e quindi un block cipher darebbe lo stesso ciphertext per quei blocchi (un intruder potrebbe sfruttare questo a suo vantaggio per capire il plaintext). Per risolvere questo possiamo introdurre della "randomicità" nella creazione del ciphertext, in modo che 2 blocchi di plaintext uguali siano criptati in maniera diversa, attraverso il **CBC** (Cipher Block Chaining): A genera una stringa random di k-bit (Inizialization Vector **[IV]**, che denoteremo con $c(0)$) e lo manda a B come plaintext. Poi sempre A manda crea il 1° ciphertext facendo XOR tra 1° plaintext $[m(1)]$ e $c(0)$ e poi criptando tutto con la chiave condivisa, ovvero $c(1) = K_S(m(1) \oplus c(0))$. E così via, cioè quando A vuole mandare il blocco di plaintext $m(i)$ crea il relativo ciphertext da mandare a B con $\boldsymbol{c(i) = K_S(m(i) \oplus c(i-1))}$. Per recuperare il messaggio di plaintext B fa un altro XOR (proprietà dell'XOR), ovvero:

$$\boldsymbol{m(i) = (m(i) \oplus c(i-1)) \oplus c(i-1) = m(i) \oplus (c(i-1) \oplus c(i-1))}$$

B. **CHIAVE ASIMMETRICA** (o **PUBBLICA**) → ci sono 2 chiavi: la chiave pubblica è nota sia ad A sia a B, mentre la chiave privata è nota solo ad A o solo a B (chiameremo K_B^+ = chiave pubblica di B, e K_B^- = chiave privata di B). Quindi A cripta il plaintext (m) in ciphertext usando la chiave pubblica di B e un algoritmo di criptazione noto (ovvero fa $\boldsymbol{K_B^+(m)}$); B riceve il messaggio e, tramite la sua chiave privata e un algoritmo di decriptazione, ottiene il plaintext originale, ovvero $\boldsymbol{K_B^-\left(K_B^+(m)\right) = m}$ [si vede poi che fare $K_B^-\left(K_B^+(m)\right) = m$ oppure $K_B^+\left(K_B^-(m)\right) = m$ dà lo stesso risultato].
Questo metodo però ha **2 problemi**: la scelta della chiave pubblica e dell'algoritmo di criptazione deve essere fatta in modo che l'intruder non possa saperle, e bisogna avere una "digital signature" (firma digitale) per legare l'inviante al messaggio (altrimenti l'intruder può fingersi l'inviante).
L'algoritmo principale di chiave pubblica è **RSA** (che funziona grazie alle proprietà matematiche dell'operatore modulo %), dove per generare K_B^+ e K_B^-:
   - Si scelgono 2 numeri primi grandi (p e q), e si fa $n = pq$ e $z = (p-1)(q-1)$;
   - Si sceglie un numero e ($< n$) che non ha fattori comuni con z (numeri primi "relativi"), che poi useremo nella fase di criptazione ($e =$ *encrypting*);
   - Si sceglie un numero d tale che $ed \,\%\, z = 1$, che useremo nella decriptazione ($d =$ decrypting);
   - Da cui $\boldsymbol{K_B^+ = (n, e)}$ e $\boldsymbol{K_B^- = (n, d)}$;
   - Il ciphertext (c) con cui A invia il suo plaintext (m) è dato da $\boldsymbol{c = m^e \,\%\, n}$, che verrà decriptato da B con $\boldsymbol{m = c^d \,\%\, n}$.

⚠ RSA è usato spesso in combinazione con la crittografia a chiave simmetrica!

2) INTEGRITÁ e FIRMA DIGITALE

Partiamo dalle **FUNZIONI DI HASH** (ovvero funzioni che prendono un input m e tirano fuori una stringa di dimensione fissa $H(m)$ chiamata "hash"; hanno la proprietà che, dati 2 input diversi x e y, è garantito che $H(x) \neq H(y)$ [garantisce che l'intruder non possa sostituire un messaggio protetto da hash con un altro]). Per avere **INTEGRITÁ** sul messaggio, oltre ad usare Hash, A e B devono condividere una stringa segreta di bit, ovvero la CHIAVE DI AUTENTICAZIONE (s) con la seguente procedura:

- A crea il plaintext m, lo concatena con s e ne fa hash, ottenendo il **MAC** (Message Authentication Code), ovvero $\boldsymbol{H(m + s)}$;
- A aggiunge il MAC ad m, creando un messaggio esteso $\boldsymbol{(m, H(m + s))}$ che manda a B;
- B riceve il messaggio esteso, che vede come $\boldsymbol{(m, h)}$ e, conoscendo s, calcola anche lui il MAC $H(m + s)$: dunque se $\boldsymbol{H(m + s) == h}$, B vede che è tutto ok!

Parlando invece della **FIRMA DIGITALE** ("digital signature"), questa è una tecnica crittografica che ci permette di autenticare la nostra identità. La procedura è la seguente (partendo da B che vuole firmare un documento m):

- B usa la sua chiave privata per firmare, ovvero la sua firma del documento è $\boldsymbol{K_B^-(m)}$;
- Per dimostrare che il documento è stato firmato veramente da B, A applica la chiave pubblica di B (K_B^+) (perché non ha la chiave privata di B) e per la proprietà vista prima, $\boldsymbol{K_B^+\left(K_B^-(m)\right) = m}$, quindi si riottiene il documento originale, tutto ok!

⚠ Si nota anche subito che la firma digitale garantisce INTEGRITÁ, in quanto se si ha un documento modificato m', questo non combacerà con quello ottenuto dalla firma digitale!

⚠ Nella pratica si usa l'hash del documento e non il documento stesso perché più piccolo (quindi meno potenza computazionale richiesta), ovvero B firma $\boldsymbol{K_B^-\left(H(m)\right)}$ [e A fa ovviamente la comparazione con $H(m)$]!

Quindi il MAC non richiede nessuna criptazione (né chiave pubblica né chiave simmetrica), mentre la firma digitale ha bisogno dell'hash e di criptare con la chiave privata (per poi decriptare con chiave pubblica) [quindi **più pesante** computazionalmente].

Un'importante applicazione della firma digitale è la certificazione di chiave pubblica, usata per legare una chiave pubblica ad un'entità; questo viene fatto con la **CA** (Certification Authority) con procedura:

- CA verifica che un'entità (persona, router, etc…) è chi dice di essere;
- Quando CA ha verificato l'identità, crea un certificato che lega la chiave pubblica dell'entità a quella identità (certificato firmato direttamente dalla CA).