HASH E CODICI PER AUTENTICAZIONE DEI MESSAGGI

Durante la comunicazione bisogna anche garantire:

I) INTEGRITÀ: Il dato non deve essere modificato, se lo è bisogna rilevare la modifica.

II) AUTENTICITÀ: Il dato deve provenire da fonte certa.

GARANTIRE L'INTEGRITÀ:

Per garantire (unicamente) l'integrità, un metodo semplice è utilizzare un DIGEST.

DIGEST: hash ricavato dal dato originale (di piccole dimensioni).

Se il destinatario conosce il digest, può ricalcolarlo e confrontarlo con il dato ricevuto ed assicurarsi quindi che il dato sia integro.

GARANTIRE L'AUTENTICITÀ

Per una architettura crittografica simmetrica in cui entrambi gli end-point hanno la chiave:

I) Il mittente genera un TAG crittografico usando il dato da trasmettere, la chiave ed una funzione AUTHENTICATE.

Trasmette il messaggio ed il tag.

II) Il destinatario verifica il tag con la chiave, il messaggio ricevuto e la funzione VERIFY che vanno FALSE in caso di messaggio alterato.

In questa architettura però il destinatario verifica solo che il mittente possiede la chiave segreta ma non può distinguere i mittenti, ne sapere chi sono.

Se l'attaccante ruba la chiave sarà impossibile identificarlo.

FUNZIONI DI HASH

Le funzioni di hash permettono di garantire l'integrità in 2 modi:

I) NON CRITTOGRAFICI: garantire l'integrità dei dati contro guasti / errori dovuti a malfunzionamenti, errori di trasmissione, ecc.

II) CRITTOGRAFICI: nonostante gli algoritmi siano pubblici non è possibile per attaccanti razionali manipolare i dati.

Le funzioni hash mappano stringhe binarie di dimensioni arbitrarie a stringhe binarie ~~fisse~~ di lunghezza fissa N.

Se l'input cambia allora il digest sarà solamente diverso mentre a parità di input il digest è sempre lo stesso.

Non è possibile garantire che ogni input generi digest diversi in assoluto ma la funzione hash non permette di invertire la relazione dato → digest. Non è quindi possibile produrre una COLLISIONE senza conoscere il dato che ha prodotto il digest.

Non si riesce a prevedere come una modifica ai dati viene propagata al digest, nonostante l'algoritmo sia noto, per design.

Se il design dell'algoritmo è fallato ed in qualche modo si riesce a capire come creare una collisione, la funzione non è più crittografica ma può ancora essere usata per usi non crittografici (ad esempio MD5).

La DIMENSIONE del digest è l'unico parametro che può influire sul livello di sicurezza della funzione hash (a patto che sia sicura).

Considerando che data la lunghezza n del digest è possibile produrre 2^n digest, per scegliere n si tenga conto della regola:

- Una funzione di hash garantisce una sicurezza pari a $2^{n/2}$ BIT (quindi per sicurezza 128 BIT serve hash 256 BIT)

  Questo è dovuto al fatto che l'attacco più efficiente per trovare le collisioni (birthday attack) ~~garantisce~~ dimostra limita la sicurezza contro le collisioni a $\sqrt{2^n}$ per n elementi.

MESSAGE AUTHENTICATION CODE (MAC)

Questa funzione prende in input un messaggio di dimensione arbitraria ed una chiave (simmetrica), produce un TAG di dimensione fissa.

Chi riceve può ricalcolare il tag e verificarlo per verificare l'autenticità dei messaggi. Non fornisce informazioni sul mittente, solo che ha la chiave.

L'attacco più efficiente contro il MAC è un brute-force sulla chiave.

Il livello di sicurezza del MAC è dato in gran parte dalla lunghezza della chiave ma anche dalla lunghezza del tag e dalla struttura interna dell'algoritmo.

Teoricamente è possibile troncare il tag per comprimere l'overhead delle comunicazioni ma conviene non farlo (potrebbe compromettere la sicurezza).

Poiché il mittente non è autenticato, attacchi come il REPLY ATTACK (un messaggio legittimo viene copiato intercettato e ritrasmesso da un attaccante) possono facilmente avere successo.

Una strategia è aggiungere dei TAGS al messaggio originale su cui poi fare il MAC, in questo modo ci sono informazioni di servizio che consentono di bloccare (riconoscere) questi attacchi.

REFLECTION ATTACK: un messaggio viene riflesso dall'attaccante al mittente in caso di comunicazione full-duplex.

Le funzioni MAC NON sono PRIMITIVE, quindi devono essere costruite su altre primitive, per esempio su f. hash (HASH MAC)

È importante usare algoritmi a tempo costante (tempo di esecuzione non dipende da input) per calcolare il MAC per evitare TIMING ATTACK, cioè attacchi basati sul tempo di esecuzione della funzione.