Funzioni di HASH

Funzioni Hash: caratteristiche

- Una **funzione** di **Hash** (o **Message Digest**) è una funzione **H** che, dato un **input M** di dimensione qualsiasi, produce un output **Y** (**Hash**) di dimensione fissa (in genere 128-160-256-512 bit): **Y = H(M)**.
- L'idea alla base è che il valore hash **H(M)** sia una **rappresentazione non ambigua** e **non falsificabile** del messaggio **M**.
- Dato un input **X** deve essere **altamente improbabile** che un altro input **Z ≠ X** generi lo stesso hash (**Collisioni**) cioè che:

  H(Z) = H(X).
- **One-Way** (**irreversibili**): cioè deve essere computazionalmente impossibile, noto **H(M)**, ricalcolare **M**.
- Inoltre la funzione deve essere **computazionalmente** semplice da eseguire, cioè calcolare **Y** come **H(M)**.

Funzioni Hash: collisioni

Le **funzioni Hash** si **classificano** in base a due proprietà che rappresentano la **resistenza alle collisioni**:

- **Resistenza Debole alle Collisioni**: Dato M, è computazionalmente impraticabile trovare un altro M´, diverso da M, tale che **H(M) = H(M´)**.

- **Resistenza Forte alle Collisioni**: Computazionalmente impraticabile trovare una coppia (M, M'), con M' ≠ M, tale che **H(M) = H(M')**.

Dalla **Resistenza alle Collisioni** dipende la **sicurezza** di una funzione Hash!

Funzioni Hash: attacchi

Le **due** proprietà sull'**assenza** di **collisioni** (**debole** e **forte**), corrispondono **due diversi tipi di attacchi** :

- **Attacco a forza bruta** (**Brute Force**):

  Trovare un **messaggio** che produca un **dato hash** (cioè un **hash** uguale a quello di un **messaggio dato**).

- **Attacco del compleanno** (**Birthday Attack**):

  Trovare **due messaggi diversi** che producano lo **stesso hash**, **indipendentemente** dal valore di questo **hash**.

MD5

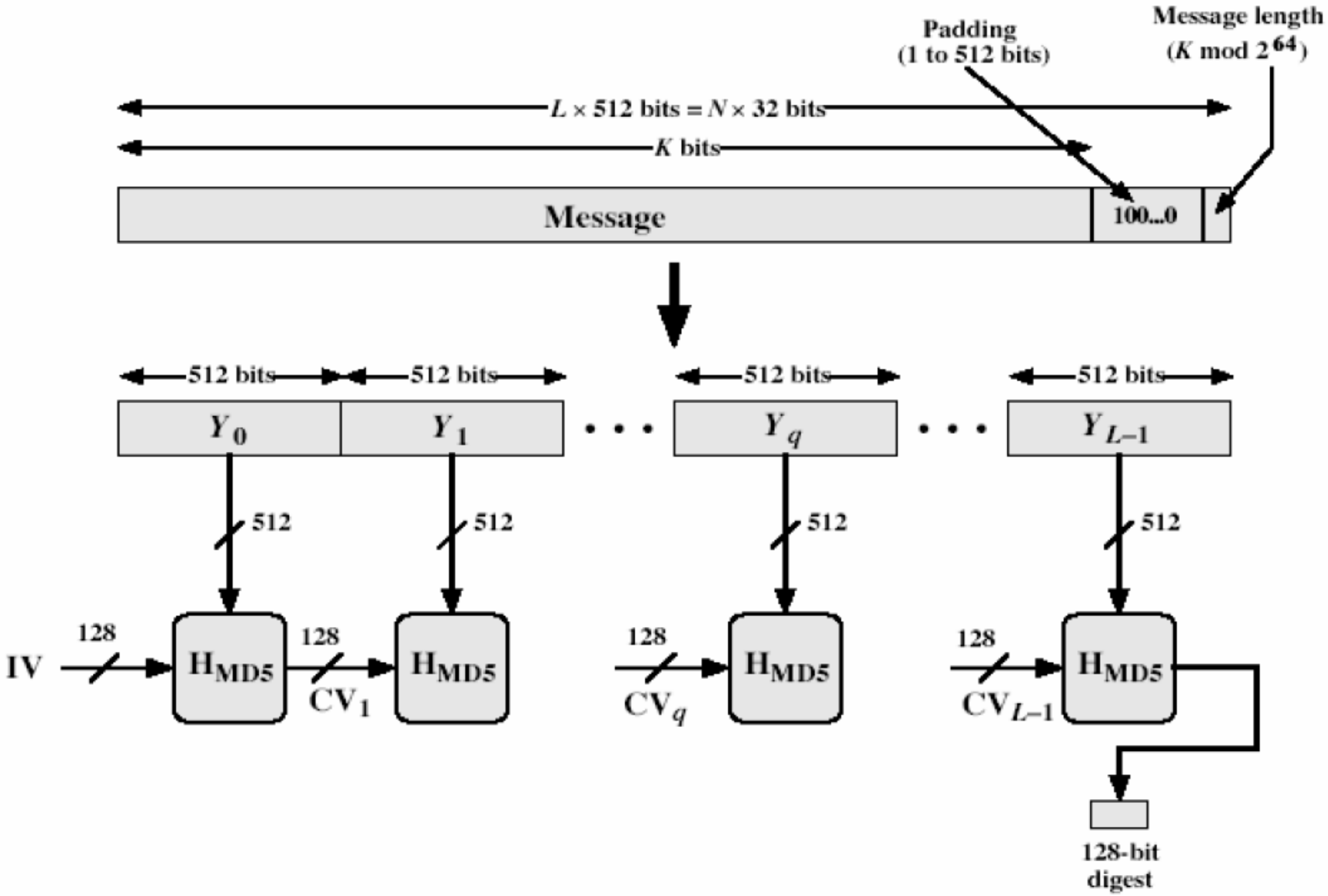
- L’algoritmo di sintesi **MD5** (**Message Digest - Rivest** 1992) è stato uno dei più usati algoritmi hash fino a qualche anno fa.
- L’algoritmo produce una **stringa di 128 bit** (32 caratteri esadecimali) a partire da un **messaggio** di lunghezza qualsiasi, con un’elaborazione piuttosto complicata in modo che ogni bit in uscita dipenda da ogni bit in ingresso.
- Il messaggio da **sintetizzare**, visto come una stringa di bit, viene diviso in **blocchi di 512 bit**; l’ultimo blocco deve essere completato con bit aggiuntivi e contiene negli ultimi 64 bit la **lunghezza originale del messaggio**.
- Viene usato un buffer di 128 bit, inizializzato a un valore **prefissato**; l’algoritmo mescola completamente ogni blocco di 512 bit con il buffer di 128 bit, attraverso un procedimento in **4 fasi** (di **16 passi ciascuna**); al termine il buffer contiene la **sintesi** del **messaggio** (**digest**).

Algoritmo MD5 - 1

- La lunghezza del messaggio è resa pari a 448 mod 512 aggiungendo un bit 1 e poi tanti 0 (**padding**).
- Viene poi aggiunta una stringa di 64 bit, contenente la **lunghezza** del **messaggio originale** (si ottiene così una stringa multipla di 512).
- Viene inizializzato il **buffer** con 4 variabili da 32 bit l'una (8 cifre esadecimali) che conterranno poi l'uscita.
- I **valori** di **inizializzazione** del buffer sono:

  A=67452301, B=EFCDAB89, C=98BADCFE, D=10325476
- Elabora il messaggio in blocchi da **512 bit**.
- Utilizza **4 fasi** (di **16 passi** ciascuna), in cui sono svolte operazioni **bitwise** sul **blocco** di **messaggio** e sul **buffer**.
- Il **codice hash** è il **contenuto** del **buffer** alla **fine** dell'**algoritmo**.

Algoritmo MD5 - 2



Applicazioni Pratiche MD5 - 1

La **crittografia** tramite algoritmo **MD5** viene applicata in tutti i settori dell'informatica che lavorano e che trattano dati sensibili.

Ad esempio, viene utilizzata per controllare che uno **scambio** di dati sia avvenuto **senza perdite** (**integrità**) , semplicemente attraverso il **confronto** del **digest prodotto** dal **file inviato** con quello prodotto dal **file ricevuto**.

È diffuso anche come supporto per l'**autenticazione** degli utenti attraverso i linguaggi di scripting web server-side (**PHP** in particolare): durante la registrazione di un utente su un portale internet, la **password** scelta durante il processo verrà **codificata** tramite **MD5** e **memorizzata** nel **database**.

Successivamente, durante il **login** la **password** immessa dall'utente subirà lo **stesso trattamento** e verrà confrontata con la copia in possesso del server per avere la certezza dell'**autenticità** del login.

```
<?php
    $my_var = 'Questa è la stringa che voglio codificare';  //Assegno ad una variabile la stringa che voglio codificare con l'algoritmo MD5
    echo md5($my_var);  //Stampo a video la codifica MD5 della stringa memorizzata nella variabile
?>
```

In C#

Classe MD5 contenuta nel namespace **System.Security.Cryptography**

```
MD5 md5Hash = MD5.Create("TextToHash")
```

Applicazioni Pratiche MD5 - 2

Molte volte la password viene inserita nel database con opportuna **cifratura simmetrica** (per **poterla decifrare**) ed il controllo con la codifica **MD5** viene eseguito **concatenando** la **password** con un **token** che il **server web** invia all'utente che si vuole autenticare.

Il **client** codifica con **MD5** il **token concatenato** con la **password** e lo invia al server (in modo che la password **non viaggi** in **chiaro** e **non** viaggi sempre la **stessa password codificata**) ed il server effettua lo stesso controllo all'atto della ricezione.

SHA (Secure Hash Algorithm)

Sono stati **sviluppati** dalla **National Security Agency** (**NSA**) e **pubblicati** dal **National Institute of Standards and Technology** (**NIST**).

Gli algoritmi nascono come modifiche del **MD5** e sono suddivisi in quattro categorie:

- **SHA-0** fu sviluppato dal NIST e dall' NSA nel 1993 (obsoleto).
- Fu poi rivisto nel 1995 e chiamato **SHA-1**. Entrambi producono codici hash di **160 bit**.
- Usano come buffer 5 variabili di 4 byte ciascuna (32x5=160) , inizializzate a valori prefissati.
- È stato l'algoritmo hash maggiormente preferito.
- Successivamente nacque la nuova versione, la **SHA-2**, che fu suddivisa in diverse famiglie a seconda della **lunghezza** in bit del **codice hash**: SHA-256, SHA-384, SHA-512. La struttura di tali algoritmi è simile a quella di **SHA-1**.
- **SHA-3** è l'ultimo membro della famiglia di standard, rilasciato dal **NIST** il 5 agosto 2015 (SHA3-224, SHA3-256, SHA3-384 e SHA3-512).

Algoritmo completamente diverso dai predecessori.

Sono usati in **SSL/TLS**.

In C# esistono classi SHA1, SHA256, SHA512.

HMAC
(Hash Message Authentication Code)

É una modalità per l'**autenticazione di messaggi** (**message authentication code**) basata su una funzione di hash, utilizzata in diverse applicazioni legate alla **sicurezza informatica**.

Tramite **HMAC** è infatti possibile garantire sia l'**integrità**, sia l'**autenticità** di un **messaggio**.

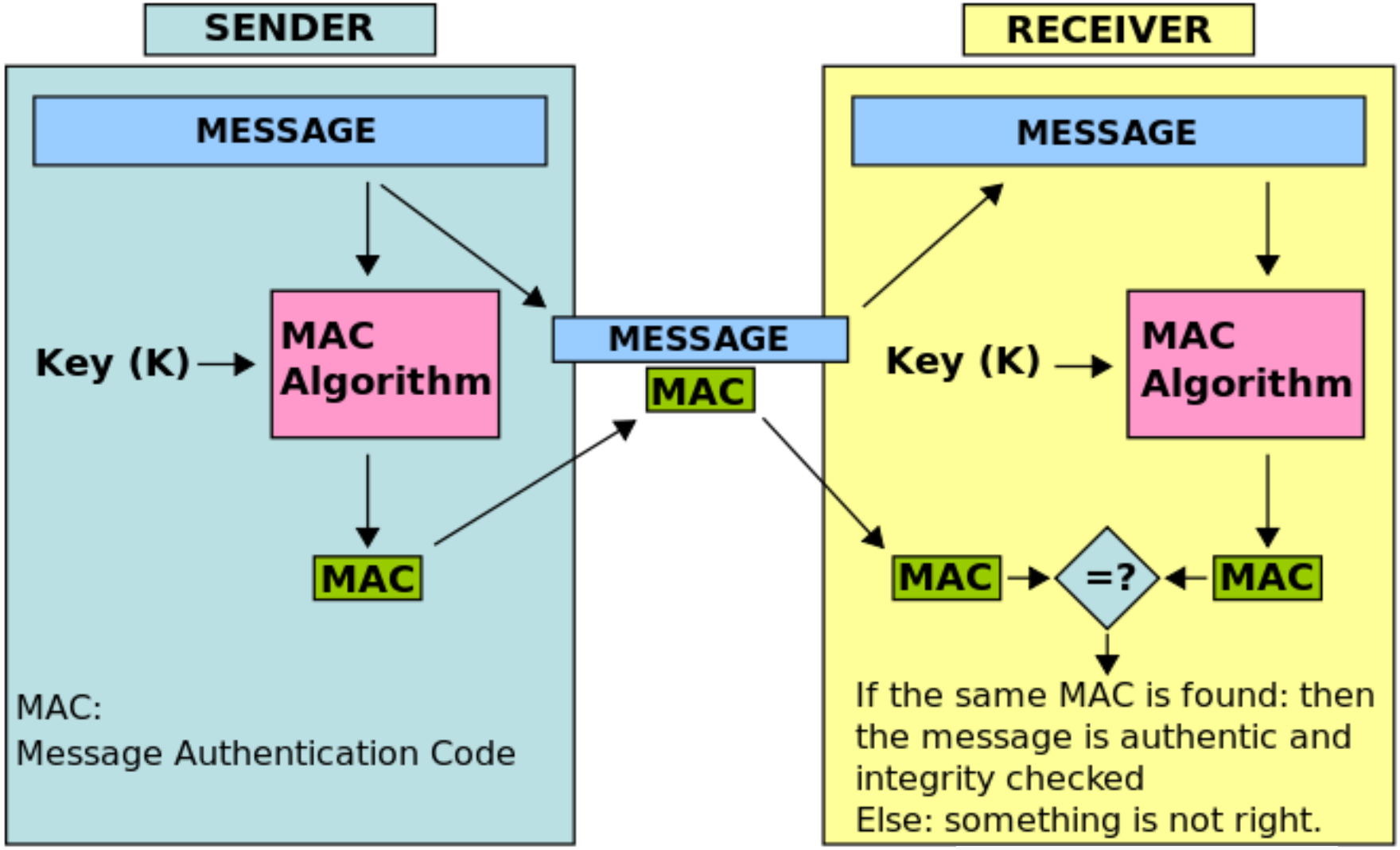
HMAC utilizza infatti una **combinazione** del **messaggio originale** e una **chiave segreta** per la **generazione** del **codice hash**.

Una caratteristica peculiare di **HMAC** è il non essere legato a nessuna funzione di **Hash** in particolare, questo per rendere possibile una sostituzione della funzione nel caso non fosse abbastanza sicura.

Nonostante ciò le funzioni più utilizzate sono **MD5** e **SHA-1**, entrambe attualmente considerate poco sicure.

È **obbligatorio** per la **sicurezza IP** (**IPsec**) usata nelle **VPN** e viene usato in **protocolli internet** come **SSL/TLS.** In questi protocolli ogni messaggio è firmato con un cosiddetto codice di autenticazione dei messaggi , o in breve **MAC**. Se mittente e destinatario sono d'accordo su una **chiave** e un codice di **hashing**, il destinatario può verificare che il messaggio provenga dal mittente e che lo stesso messaggio non sia stato modificato.

HMAC



qualcosa non quadra!

Un utente malintenzionato può modificare il messaggio ma **non** conosce la **chiave**. **Non può calcolare** il MAC corretto ed il destinatario saprà che il messaggio non è autentico.