## Resistenza alle collisioni

Una collisione avviene quando input differenti producono lo stesso identica *hash*. Quindi, una funzione di hash è considerata **resistente alle collisioni** fino a quando qualcuno trova una collisione. È da notare che le collisioni esisteranno sempre per qualsiasi funzione di hash, siccome i possibili input sono infiniti, mentre i possibili output sono limitati.

In altre parole, una funzione di hash è resistente alle collisioni quando la **possibilità di trovare una collisione è così bassa che richiederebbe milioni di anni di computazioni**. Quindi, anche se non esistono funzioni di hash resistenti alle collisioni, alcune sono abbastanza forti da essere considerate resistenti (ad es., *SHA-256*).

## Resistenza alla preimmagine

La proprietà di resistenza alla preimmagine è legata al concetto di funzioni unidirezionali. Una funzione di hash è considerata resistente alla preimmagine quando c'è una **probabilità molto bassa che qualcuno trovi l'input che ha generato un determinato output**. In questo caso, un attaccante dovrebbe provare a indovinare l'input esaminando un determinato output. Una collisione, invece, avviene quando qualcuno trova due input differenti che generano lo stesso output, ma non importa quali input vengono usati.

La proprietà di resistenza alla preimmagine è importante per la **protezione dei dati** perché un semplice *hash* di un messaggio può dimostrarne l'autenticità, senza dover rivelare le informazioni.

## Resistenza alla seconda preimmagine

È una via di mezzo tra le altre due proprietà. Un attacco alla seconda preimmagine avviene quando qualcuno è in grado di **trovare un input specifico che genera lo stesso output di un altro input che conoscono già**.

Questo caso comprende sempre la scoperta di una collisione e di conseguenza, qualsiasi funzione di hash resistente alle collisioni è anche resistente agli attacchi alla seconda preimmagine.

## APPLICAZIONI DI RILIEVO

- **Verifica dell'integrità di un messaggio**: per mezzo delle funzioni hash è possibile determinare, ad esempio, se sono state compiute modifiche ad un messaggio (o ad un file) confrontando il suo hash prima e dopo la trasmissione. Un particolare uso ne viene fatto nella maggior parte degli algoritmi di firma digitale.
- **Verifica delle password**: nelle applicazioni che necessitano di un'adeguata autenticazione è troppo rischioso memorizzare le password di tutti gli utenti in chiaro, cioè su un file non cifrato, soprattutto nel caso in cui quest'ultimo venga compromesso. Un modo per evitare di andare incontro a una vera e propria violazione della sicurezza è quello di memorizzare solo il valore dell'*hash* di ogni password.

## ALGORITMI DI HASHING FAMOSI

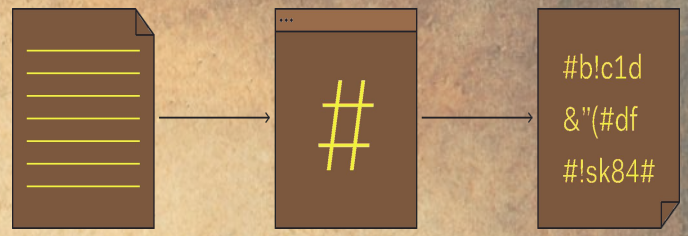
| Nome | Lunghezza (bits) |
|---|---|
| MD4 | 128 |
| MD5 | 128 |
| SHA-1 | 160 |
| SHA-256 | 256 |
| SHA-512 | 512 |

## RIFERIMENTI


➢ https://www.c3t.it/projects/awareness/articoli&brochure/hashing/

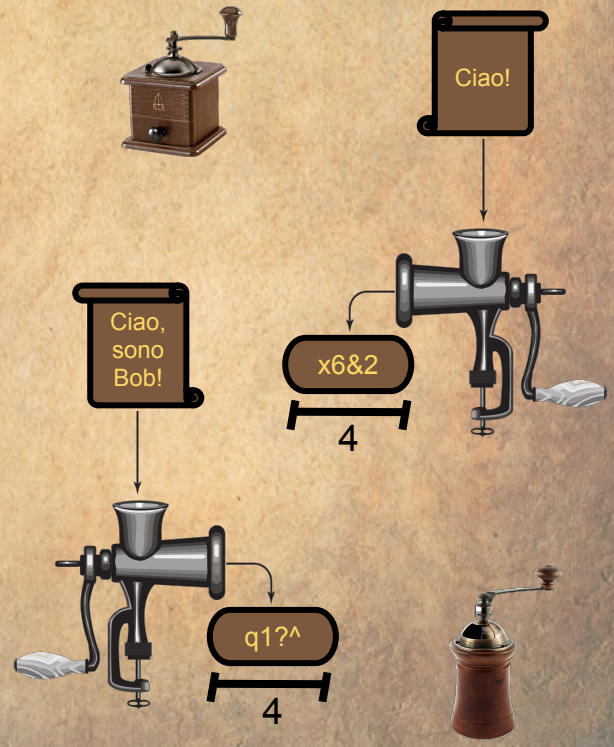


*POR FESR 2014-2020 obiettivo Crescita e Occupazione (CreO)*


# TECNICHE DI HASHING

Scopri cosa sono le tecniche di hashing, come funzionano e qual è la loro importanza nell'ambito digitale.



## COS'È L'HASHING?

Con **hashing** si intende il processo che genera un output di dimensione fissa partendo da un input di dimensione variabile. Questo viene fatto attraverso l'uso di formule matematiche conosciute come **funzioni di hash**.

Non tutte le funzioni hash implicano l'uso di crittografia; se lo fanno, tali funzioni, prendono il nome di **funzioni crittografiche di hash**.

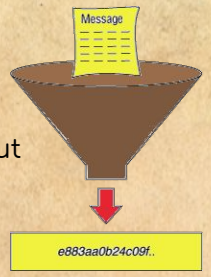
Si può vedere l'hashing come un modo di fare il "riassunto informatico" di un documento digitale.

Le funzioni di hash sono **deterministiche**. Questa proprietà significa che fino a quando l'input non cambia, l'algoritmo di hashing continuerà a produrre lo stesso output (conosciuto anche come *digest* o *hash*).

In genere, gli algoritmi di hashing sono progettati come **funzioni unidirezionali**, ovvero che non possono essere facilmente invertite senza enormi quantità di risorse e tempo di computazione. In altre parole, è piuttosto semplice creare l'output dall'input, ma relativamente difficile andare nella direzione opposta (generare l'input partendo dal solo output). In linea di massima, più è difficile trovare l'input, più l'algoritmo di hashing è considerato sicuro.

## COSA FANNO LE FUNZIONI DI HASH?

Le funzioni di hash fanno **complesse operazioni matematiche** sulla stringa di input fino a renderla "irriconoscibile" e portarla ad avere una dimensione di output predefinita. Il miglior modo per concepire una funzione di hash è quello di vederla come una "macchinetta" che prende in ingresso una stringa di qualsiasi dimensione e ne tira fuori un'altra di dimensione fissata.




## QUAL È LA LORO IMPORTANZA?

Le funzioni di hash convenzionali hanno una vasta gamma di casi d'uso, tra cui ricerca in database, analisi di grandi file e gestione dei dati. Invece, le **funzioni crittografiche di hash** vengono usate estensivamente nelle applicazioni per la sicurezza informatica, come l'**autenticazione di messaggi**, l'**impronta digitale**, la **firma digitale** e molte altre.

Il vero potere dell'hashing si vede quando si ha a che fare con enormi quantità di informazioni. Per esempio, possiamo elaborare un grande file o un dataset attraverso una funzione di hash e usare il suo output per **verificare velocemente l'accuratezza e l'integrità dei dati**. Questo è possibile grazie alla natura deterministica delle funzioni di hash: l'input risulterà sempre in un **output semplificato e condensato** (*hash*). Questa tecnica rimuove quindi la necessità di dover archiviare e "ricordare" grandi quantità di dati.

## ESEMPI DI FUNZIONI DI HASH

Diverse funzioni di hash produrranno output di dimensione differente, ma la **dimensione dell'output per ciascun algoritmo di hashing sarà sempre costante**. Per esempio, l'algoritmo *SHA-256* produce output di 256 bit (64 caratteri nel formato esadecimale), mentre l'algoritmo *SHA-1* genera *digest* di 160 bit (40 caratteri in esadecimale).

Per illustrare il concetto, si può notare che elaborando le parole "C3T" e "c3t" attraverso l'algoritmo di hashing *SHA-256* otteniamo 2 *digest* di **lunghezza fissa ma completamente diversi**.

| Testo | Digest SHA-256 (64 caratteri) |
|---|---|
| C3T | 1bd0711188be53fa0b57eb4e464c03d529c08649a813b0c370863aa04671ad86 |
| c3t | cf8215a98b29de28d5dcc09288bee5bcd345eb742ad86718e7923ccbe0edc0f2 |

Similmente con l'algoritmo *SHA-1* otteniamo due *digest* di 40 caratteri.

| Testo | Digest SHA-1 (40 caratteri) |
|---|---|
| C3T | dff8c1d3213a8bae294af41f16cd3754bbec0919 |
| c3t | e64c311f67f5cd0751012c8100a6fab1a937039e |

Un piccolo cambiamento nel testo di input risulta in un valore di hash totalmente differente. Inoltre, fissato l'algoritmo, gli output avranno sempre una dimensione fissa, a prescindere dalle dimensioni dell'input.

## FUNZIONI CRITTOGRAFICHE DI HASH

Una funzione di hash che utilizza tecniche crittografiche può essere definita come una **funzione crittografica di hash**. In generale, rompere una funzione crittografica di hash richiede una miriade di **tentativi a forza bruta**. Per riuscire a "invertire" una funzione crittografica di hash, è necessario indovinare l'input a suon di tentativi fino a quando viene prodotto l'output corrispondente. Tuttavia, c'è anche la possibilità che diversi input possano produrre lo stesso esatto output, caso in cui avviene una "**collisione**".

Tecnicamente, una funzione crittografica di hash deve seguire tre proprietà per essere considerata effettivamente sicura:

- **Resistenza alle collisioni**: computazionalmente intrattabile trovare una coppia di input distinti che producono lo stesso hash come output.
- **Resistenza alla preimmagine**: computazionalmente intrattabile "invertire" la funzione di hash (trovare l'input partendo da un dato output).
- **Resistenza alla seconda preimmagine**: computazionalmente intrattabile trovare un secondo input in collisione con un input specifico.