Agenzia per la
Cybersicurezza Nazionale

LINEE GUIDA FUNZIONI CRITTOGRAFICHE

Funzioni di Hash

MAGGIO 2026

Versione	Data di pubblicazione	Note
1.0	**07/12/2023**	**Prima pubblicazione**
1.1	**12/05/2026**	**Aggiunta sezione “Scopo del documento”, ulteriori modifiche minori**

Sommario

Indice delle figure

Indice delle tabelle

Scopo del documento

Questo documento costituisce parte della serie "Linee Guida Funzioni Crittografiche" e fornisce le raccomandazioni di ACN in merito alle **funzioni di hash**, da adottare in tutti i contesti in cui si necessita di garantire l'integrità di dati inviati o salvati. Per ulteriori informazioni sulle Linee Guida e sulle utilità delle funzioni crittografiche in base ai contesti specifici, si faccia riferimento al documento introduttivo della serie [1].

Ogni documento tiene in considerazione le minacce presenti al giorno della sua pubblicazione. Data la diversa natura dei sistemi informativi di destinazione, non è possibile garantire che queste raccomandazioni possano essere utilizzate senza adattamenti specifici. In qualsiasi caso, la pertinenza dell'attuazione delle soluzioni proposte deve essere sottoposta, preventivamente, a valutazione e validazione da parte dei responsabili della sicurezza dei sistemi informativi di destinazione.

I contenuti delle Linee Guida sono indirizzati a sviluppatori, produttori di dispositivi e fornitori di servizi digitali, al fine di promuovere l'utilizzo di primitive crittografiche e relativi parametri di configurazione sicuri fin dalla fase di progettazione di prodotti, reti, applicazioni e servizi. Questi documenti sono altresì rivolti a security manager, referenti e responsabili della cybersicurezza di soggetti pubblici e privati affinché verifichino che i sistemi utilizzati dalla propria organizzazione siano conformi alle raccomandazioni fornite.

La legge 28 giugno 2024, n. 90 relativa a "Disposizioni in materia di rafforzamento della cybersicurezza nazionale e di reati informatici", all'articolo 9, stabilisce a tal fine che *«le strutture di cui all'articolo 8 della presente legge nonché quelle che svolgono analoghe funzioni per i soggetti di cui all'articolo 1, comma 2-bis, del decreto-legge 21 settembre 2019, n. 105, convertito, con modificazioni, dalla legge 18 novembre 2019, n. 133, e al decreto legislativo 18 maggio 2018, n. 65, verificano che i programmi e le applicazioni informatiche e di comunicazione elettronica in uso, che utilizzano soluzioni crittografiche, rispettino le linee guida sulla crittografia nonché quelle sulla conservazione delle password adottate dall'Agenzia per la cybersicurezza nazionale e dal Garante per la protezione dei dati personali e non comportino vulnerabilità note, atte a rendere disponibili e intellegibili a terzi i dati cifrati»*.

Il documento è stato curato dal Centro Nazionale di Crittografia istituito presso ACN.

Lista dei simboli matematici utilizzati

$\{0,1\}$	Campo binario dei valori assumibili da un singolo bit	$\|\|$	Concatenazione di stringhe
$\{0,1\}^n$	Spazio vettoriale delle stringhe binarie di lunghezza n	$\oplus$	Operazione XOR, cioè somma bit a bit tra stringhe binarie
$\{0,1\}^*$	Insieme di stringhe binarie di lunghezza arbitraria	$\ll k$	Rotazione a sinistra di k posizioni con reinserimento
$O(n)$	Notazione O-grande	$\boxplus$	Somma tra elementi nel campo con $2^{32}/2^{64}$ elementi

1 Introduzione

Le funzioni di hash crittografiche sono uno strumento fondamentale per la crittografia moderna poiché, grazie alle loro proprietà, rendono possibile la verifica dell'integrità dei dati, trovando numerose applicazioni in ambito informatico. Al fine di utilizzare un algoritmo di hash in modo sicuro e corretto, è necessario che queste proprietà fondamentali vengano rispettate. Da anni, la comunità scientifica studia le funzioni di hash al fine di assicurarsi che le più diffuse siano resistenti ad attacchi che potrebbero inficiarne la sicurezza. Alcuni enti internazionali di normazione tecnica hanno stabilito degli standard e aggiornato quelli divenuti obsoleti negli anni in seguito alla scoperta di rilevanti debolezze e vulnerabilità. Tuttavia, tali aggiornamenti non sempre sono stati recepiti dagli addetti ai lavori, portando a un utilizzo ancora diffuso di funzioni di hash ritenute compromesse.

Questo documento fornisce indicazioni e raccomandazioni sulle funzioni di hash attualmente ritenute più sicure e allo stesso tempo più efficienti.
Il documento presenta la seguente struttura: nel capitolo 2 si presentano le funzioni di hash in generale, descrivendo le proprietà di sicurezza richieste, le applicazioni crittografiche e le principali costruzioni utilizzate nella loro progettazione; nel capitolo 3 si presentano due algoritmi molto utilizzati negli ultimi anni ma considerati ormai obsoleti e quindi fortemente sconsigliati; nel capitolo 4 si introducono le due funzioni di hash raccomandate; nel capitolo 5 si riassumono in breve le caratteristiche degli algoritmi descritti nei precedenti capitoli; infine, nel capitolo 6 si richiamano brevemente le indicazioni su quali algoritmi sono raccomandati e quali sconsigliati.

2 Funzioni di hash

Una funzione di hash h è una funzione che prende in input una stringa di bit di lunghezza arbitraria e restituisce una stringa di bit di una lunghezza fissa, cioè

$$h : \{0,1\}^* \longrightarrow \{0,1\}^n.$$

L'immagine di una data stringa M, indicata con $h(M)$, è detta **digest**. Le funzioni di hash sono di grande importanza in ambito crittografico e hanno numerose applicazioni, tra cui la verifica dell'integrità dei dati e la firma digitale, descritte più nel dettaglio nella sezione 2.2. Tuttavia, al fine di utilizzarle in crittografia, è richiesto che esse soddisfino alcune importanti proprietà, analizzate di seguito.

2.1 Proprietà delle funzioni di hash crittografiche

Una funzione di hash crittograficamente sicura deve essere resistente ad attacchi da parte di terze parti, che potrebbero voler ricostruire o modificare il dato a partire dal suo digest, e quindi deve avere determinate proprietà di sicurezza. Si noti che spesso vengono aggiunte ulteriori proprietà riguardanti la computabilità e l'efficienza delle funzioni di hash. Tuttavia, qui sono riportate solo le proprietà fondamentali che devono essere garantite per ottenere una funzione di hash adatta all'uso crittografico.

2.1.1 Resistenza alla preimmagine

Una funzione di hash h si dice resistente alla preimmagine (o **one-way**) se, dato un digest $h(M)$, ricavare la stringa M risulta computazionalmente oneroso, cioè se la funzione h risulta difficile da invertire. In questo modo, un attaccante non è in grado di calcolare il dato di partenza partendo solo dal digest.

2.1.2 Resistenza alla seconda preimmagine

Una funzione di hash h si dice resistente alla seconda preimmagine se, data una stringa M_1, ricavare una seconda stringa distinta M_2 tale che $h(M_1) = h(M_2)$ risulta computazionalmente oneroso. Questo corrisponde a richiedere che sia praticamente impossibile modificare l'input mantenendo lo stesso output.

2.1.3 Resistenza alle collisioni

Una funzione di hash h si dice resistente alle collisioni se è computazionalmente oneroso trovare due stringhe distinte M_1 e M_2 tali che $h(M_1) = h(M_2)$.
Risulta evidente che, da definizione, la proprietà di essere resistente alle collisioni implica la resistenza alla seconda preimmagine, o, analogamente, trovare una seconda preimmagine implica trovare una collisione. Gli attacchi di forza bruta sulla ricerca di una collisione di una funzione di hash si basano sul cosiddetto **paradosso del compleanno** (sono chiamati infatti birthday attacks o attacchi del compleanno) e permettono di trovare una collisione con una probabilità di successo di circa il 50% dopo aver calcolato $O\left(2^{n/2}\right)$ valori di hash, dove n è la lunghezza del digest.

Minaccia quantistica

Come la maggior parte della crittografia simmetrica, le funzioni di hash risultano suscettibili agli attacchi perpetrati da un computer quantistico tramite l'**algoritmo di Grover** [2], che garantisce, tuttavia, solo un aumento quadratico della velocità degli attacchi di forza bruta. Quindi, un raddoppio delle dimensioni del digest permette di garantire lo stesso livello di sicurezza degli standard attuali, di fatto rendendo vani i miglioramenti quantistici.

2.2 Applicazioni delle funzioni di hash

Le funzioni di hash sono principalmente utilizzate per la verifica dell'integrità di un dato o per la firma digitale [3, 4]. Nel primo caso, si suppone di avere un dato M e di voler avere in ogni momento la possibilità di controllare che esso non sia stato modificato. Si può applicare a M la funzione di hash per ottenere il digest $h(M)$, che costituisce una "impronta digitale" del dato. Questa impronta può essere sfruttata in diverse situazioni: può essere tenuta segreta per verificare che il dato sia rimasto immutato semplicemente calcolando nuovamente il suo hash e confrontandolo con quello tenuto al sicuro; oppure può essere inviata, cifrata, congiuntamente al dato M, cosicché il ricevente, dopo averla decifrata, possa utilizzarla per verificare l'integrità di M come nell'esempio precedente. Un ambito di applicazione di questa tecnica è, ad esempio, la distribuzione di un software: infatti, calcolando l'hash di un aggiornamento software è possibile verificare l'integrità dell'eseguibile e il fatto che non siano presenti malware che ne alterino il contenuto.
Nel caso della firma digitale, le funzioni di hash vengono spesso utilizzate nella tecnica **hash-then-sign**: prima di firmare il messaggio, ne viene calcolato l'hash e poi la firma viene applicata solamente al suo hash. Al momento della verifica, è sufficiente calcolare l'hash del messaggio e poi verificare la firma sul digest invece che sul messaggio originale. Questa tecnica consente di risparmiare sia spazio di memorizzazione che tempo, in quanto firmare direttamente il messaggio significherebbe molto spesso doverlo prima dividere in blocchi di lunghezza adeguata e poi firmare ogni blocco singolarmente, oltre che garantire l'integrità del messaggio stesso. Per maggiori dettagli sulle firme digitali si rimanda al documento dedicato [5].
La proprietà one-way delle funzioni di hash le rende inoltre adatte nell'ambito delle funzioni di derivazione della chiave e della conservazione delle password. Le prime sono funzioni che producono chiavi a partire da una singola chiave iniziale e hanno uso, per esempio, nei terminali POS (Point Of Sale). Il **password hashing** è una pratica utilizzata per conservare le password in modo sicuro all'interno di un archivio. Per le raccomandazioni su funzioni di hash da utilizzare in questo contesto si consulti il documento dedicato [6].
È importante osservare che le funzioni di hash non assicurano invece l'autenticazione del messaggio, in quanto sono sprovviste di una chiave che permetta di garantire l'identità del mittente. Per ottenere tale proprietà è necessario ricorrere all'utilizzo di MAC (Message Authentication Code), per le cui raccomandazioni si rimanda al documento dedicato [7].
Risulta evidente che, per le applicazioni illustrate, le funzioni di hash devono essere pubbliche e note a tutti, perché è necessario che chiunque sia in grado di calcolare il digest dei dati ricevuti.

2.3 Tipologie di funzioni di hash

Le principali costruzioni di funzioni di hash si possono ricondurre a due tipologie: **iterate** o **a spugna** (sponge).
Nel primo caso si parte da una funzione f di compressione tale che $f : \{0,1\}^{n+k} \longrightarrow \{0,1\}^n$, dove k è un valore positivo. Le stringhe in input alla funzione vengono quindi compresse in output più piccoli e il dominio di f è più grande del suo codominio. Tali presupposti bastano per osservare che la funzione non può essere iniettiva quando k è strettamente positivo e risulta quindi non invertibile.
Le funzioni iterate agiscono solitamente in questo modo: il messaggio viene prima processato e diviso in blocchi, ai quali viene applicata ripetutamente la funzione di compressione e infine convertito nella dimensione prescritta tramite una trasformazione finale.
La principale famiglia di funzioni di hash iterate è rappresentata dalla costruzione di Merkle-Damgård [8, 9], illustrata in Figura 1.

Inizialmente, il messaggio subisce un'operazione detta **padding** che serve a renderlo della lunghezza corretta. In seguito, esso viene diviso in blocchi di lunghezza fissa che vengono dati in input a una funzione di compressione f.
La prima applicazione di tale funzione prende in input un valore iniziale fisso detto **vettore di inizializzazione** (IV) e il primo blocco M_1, mentre le applicazioni seguenti operano sull'output della funzione precedente e sul nuovo blocco M_i. Quando tutti i blocchi sono stati processati, la procedura è conclusa da un'operazione di finalizzazione, che restituisce il digest finale. Questo metodo è ampiamente diffuso in quanto garantisce che, qualora la funzione di compressione sia resistente alle collisioni, anche la funzione di hash nel suo complesso lo è.
Per quanto riguarda il secondo caso, le funzioni a spugna sono una costruzione più recente [10], nata per trovare una valida alternativa alla struttura di Merkle-Damgård. Anche in questo caso si sfrutta una funzione f, che però non è una funzione di compressione bensì una permutazione, che quindi mantiene la lunghezza dell'input cambiando solo le posizioni dei singoli bit.
Come illustrato in Figura 2, in seguito all'applicazione di un padding per ottenere la lunghezza corretta, l'input M viene suddiviso in blocchi da r bit, detto **bit-rate**, i quali vengono elaborati uno ad uno nella prima fase, detta di **assorbimento**. In questa, lo XOR ($\oplus$) di ogni singolo blocco con i primi r bit del registro viene alternato all'applicazione della permutazione f, fino ad esaurire i blocchi in input. Da osservare come il registro sia composto da una seconda parte di lunghezza c, detta **capacità**, i cui bit non vengono sommati ai blocchi ma che in seguito alla permutazione conterranno bit di informazione. Questa parte è fondamentale nel definire la sicurezza della funzione di hash in quanto questi bit continuano a essere utilizzati ma non vengono più resi pubblici.
Una volta conclusa la fase di assorbimento, si passa alla seconda fase: la **spremitura**. A differenza della fase precedente, vengono alternate l'estrazione di blocchi H_i di r bit e la permutazione f fino ad ottenere un digest della lunghezza n desiderata. Solitamente è sufficiente una singola estrazione seguita da troncamento. Si noti che, tramite questa costruzione, si può ottenere un output di qualsiasi lunghezza e, in tal caso, si parla di **Extendable Output Function** (XOF).

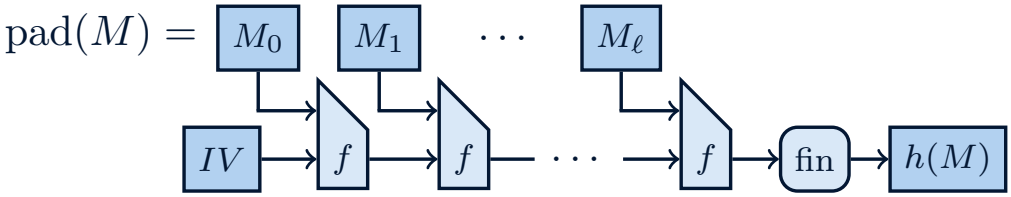

Figura 1 - Costruzione di Merkle-Damgård

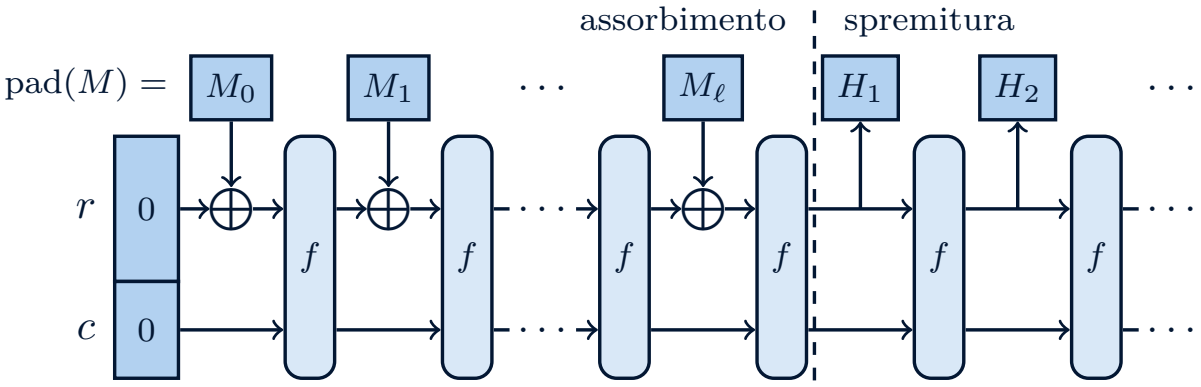

Figura 2 - Costruzione a spugna

3 Algoritmi obsoleti

Alcune funzioni di hash sono state dichiarate insicure da anni, sebbene molto spesso il loro utilizzo venga comunque mantenuto ai fini di compatibilità con i vecchi sistemi o per operazioni di verifica dell'integrità di dati ritenuti poco importanti. I due esempi più noti vengono riportati di seguito per ragioni storiche e di completezza, ma il loro uso è fortemente sconsigliato in qualsiasi applicazione, date le loro vulnerabilità.

3.1 MD5

MD5 (Message Digest) è una funzione di hash progettata da Ronald Rivest nel 1992 [11] per sostituire la precedente versione MD4. Questo algoritmo è stato ampiamente utilizzato, soprattutto nella verifica dell'integrità di file, ma la dimensione dell'output di soli 128 bit rende la funzione debole anche contro un attacco di forza bruta come quello citato nella sezione 2.1.2.
Attualmente, la funzione è considerata insicura in seguito alle collisioni individuate per la funzione di compressione e per l'intero algoritmo [12, 13].
La funzione prende un input M di al più 2^{64} bit e restituisce un digest di 128 bit ottenuto tramite una costruzione di Merkle-Damgård. La funzione di compressione f prende in input 512 bit di M e ripete per 64 round la funzione descritta in Figura 3. In essa i registri sono lunghi 32 bit e nello specifico:

- A, B, C, D vengono inizializzati con un IV fissato e nei round successivi vengono aggiornati con il risultato della funzione;
- i 512 bit di M si suddividono in 16 registri che vengono elaborati round per round tramite W_j, il quale, ogni round, assume il valore di uno di questi registri;
- K_j è una costante che cambia ogni round.

Le funzioni applicate all'interno del singolo round sono rotazione a sinistra di valore variabile ($\ll s_j$), somma modulo 2^{32} ($\boxplus$) e F_t, una funzione logica che cambia ogni 16 round.
La ricerca di collisioni in MD5 ha raggiunto complessità $O(2^{24})$ nel caso medio [14], equivalente a pochi secondi di computazione, mentre nel caso peggiore sono richieste poche ore [15].

3.2 SHA-1

SHA-1 è una funzione di hash progettata dalla NSA (National Security Agency statunitense) ed è stata la prima funzione di hash a diventare uno standard per il NIST statunitense nel 1995 [16].
La funzione prende un input M lungo al più 264 bit e restituisce un digest di 160 bit ottenuto tramite una costruzione iterata in stile Merkle-Damgård. La funzione di compressione f prende in input blocchi di 512 bit di M e ripete per 80 round la funzione descritta in Figura 4. In essa le operazioni avvengono tra registri lunghi 32 bit, detti parole, e nello specifico:

- A, B, C, D, E vengono inizializzate con un IV fissato all'inizio del primo round e nei round successivi vengono aggiornati con il risultato della funzione nel round precedente;

- i 512 bit di M si suddividono in 16 parole che vengono elaborate round per round tramite W_j, il quale assume il valore di uno di questi registri per i primi 16 round mentre dal round 17 assume valori ottenuti da uno sviluppo del messaggio (message schedule);
- K_t è una costante che cambia ogni 20 round.

Le funzioni applicate all'interno del singolo round sono rotazioni a sinistra di k posizioni ($\ll k$), somma modulo 2^{32} ($\boxplus$) e F_t, una funzione logica che cambia ogni 20 round.

In seguito alla pubblicazione di un attacco alle collisioni nel 2017 [17], è stato dimostrato che la ricerca di una collisione di SHA-1 ha una complessità inferiore a $O(2^{64})$.
Di conseguenza, SHA-1 è stato dichiarato obsoleto dal NIST nel 2011 [18] ed è stato rimosso da tutte le linee guida europee negli anni successivi tra cui SOG-IS [19], ANSSI [20], BSI [21] e ETSI [22]. Al momento, una collisione può essere trovata su un ASIC ad alte prestazioni in meno di un giorno [23].

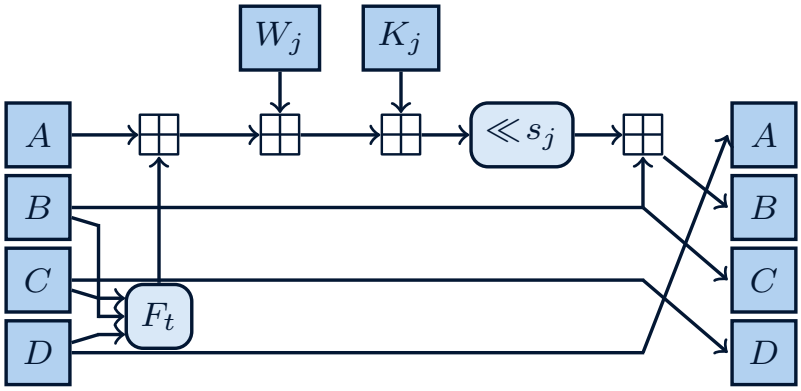

Figura 3 - Round della funzione di compressione di MD5

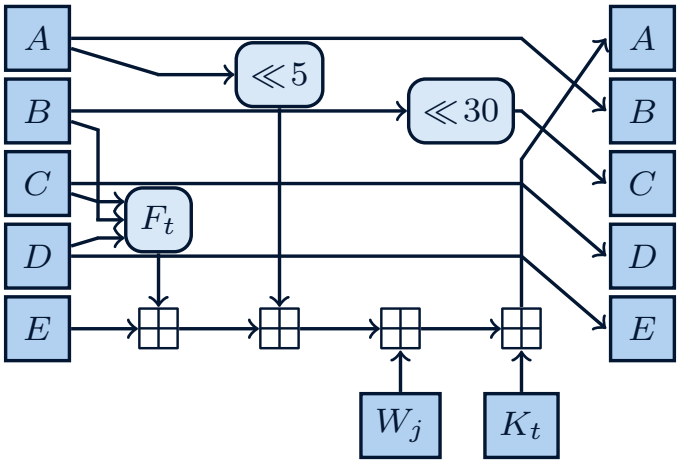

Figura 4 - Round della funzione di compressione di SHA-1

L'utilizzo di MD5 e SHA-1 è **fortemente sconsigliato**. Si raccomanda, qualora ancora in uso, di sostituirli al più presto con funzioni più moderne come quelle suggerite nel capitolo 4.

4 Algoritmi raccomandati

I due algoritmi di hash descritti in seguito sono quelli ancora considerati sicuri e standardizzati dagli organismi internazionali. Appartengono entrambi alla famiglia degli algoritmi SHA, sebbene SHA-2 sia ancora largamente più utilizzato del suo successore SHA-3.

4.1 SHA-2

SHA-2 [24, 25] è un insieme di funzioni crittografiche di hash progettato dalla NSA per migliorare le proprietà di sicurezza del predecessore SHA-1. È stato brevettato nel 2001 e reso uno standard nel 2002 [26].

Le due funzioni principali in SHA-2 prendono il nome dalla lunghezza del digest: SHA-256 e SHA-512. Similmente a SHA-1, le due funzioni prendono un input M di al più 2^{64} bit e restituiscono un output ottenuto tramite una costruzione di Merkle-Damgård. La differenza con SHA-1 risiede nella funzione di compressione f che ripete la funzione descritta in Figura 5 con:

- 512 bit di M in input, parole da 32 bit e 64 round di ripetizione per SHA-256;
- 1024 bit di M in input, parole da 64 bit e 80 round di ripetizione per SHA-512.

I registri sono simili a quelli di SHA-1:

- A, B, C, D, E, F, G, H vengono inizializzati con un IV che dipende dalla versione e nei round successivi vengono aggiornati con il risultato della funzione nel round precedente;
- il blocco di M si suddivide in 16 parole che vengono elaborate round per round tramite W_j, il quale assume il valore di una di queste parole per i primi 16 round mentre dal round 17 assume valori ottenuti da uno sviluppo del messaggio (message schedule);
- K_j è una costante che cambia ogni round.

Per quanto riguarda le operazioni svolte, a differenza di SHA-1, $\boxplus$ rappresenta la somma modulo 2^{32} oppure 2^{64} in base alla versione, e vengono introdotti i nuovi operatori logici Ch, Ma, Σ_0, Σ_1.

Oltre a SHA-256 e SHA-512, la famiglia SHA-2 comprende altre 4 funzioni di hash ottenute troncando il digest delle due funzioni principali: SHA-224, SHA-384, SHA-512/224 e SHA-512/256.

Sebbene la circostanza che SHA-2 condivida parte della sua struttura con SHA-1 abbia allarmato parte della comunità scientifica, spingendo il NIST ad aprire una nuova competizione per la ricerca di un nuovo standard per funzioni di hash, ad oggi questa famiglia di funzioni è ritenuta sicura. Infatti, per quanto riguarda gli attacchi alle collisioni, ne esistono solo alcuni a versioni piuttosto ridotte del cifrario: per SHA-512, l'attacco si ferma a 57 round su 80 ed è comparabile ad un attacco di forza bruta, mentre per SHA-256 si arriva a 52 round su 64 [27]. Risulta quindi che, al momento, non esistono attacchi in grado di inficiare la sicurezza di SHA-2, che resta un'ottima alternativa al più moderno SHA-3.

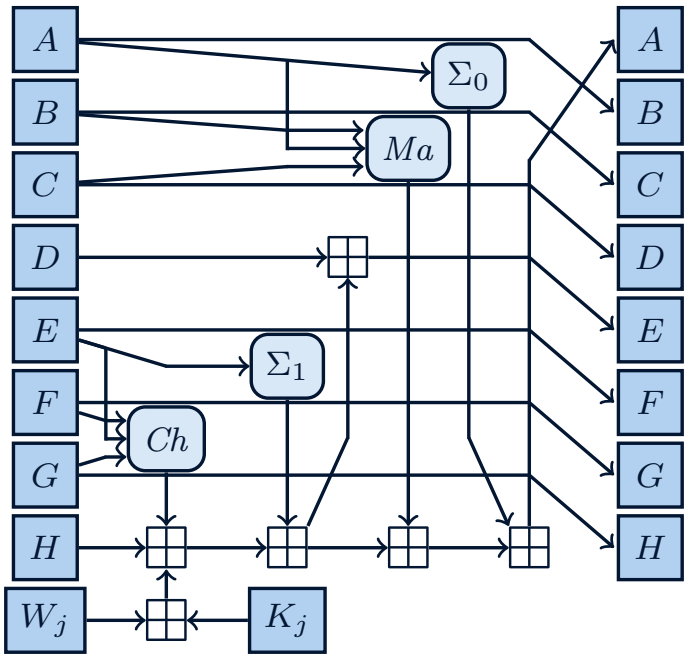

Figura 5 - Round della funzione di compressione di SHA-2

4.2 SHA-3

Nel 2006, in seguito agli attacchi teorici presentati contro SHA-1, il NIST ha indetto una nuova competizione per creare un nuovo standard per le funzioni di hash.
Keccak, una funzione di hash progettata da Guido Bertoni, Joan Daemen, Michaël Peeters e Gilles Van Assche, è stata dichiarata vincitrice della gara nel 2012, diventando così nel 2015 un nuovo standard per le funzioni di hash chiamato SHA-3 [25, 28].

SHA-3 ha una struttura totalmente diversa dai precedenti standard per le funzioni di hash essendo infatti basata su una costruzione a spugna. In particolare, i parametri di SHA-3 sono:

- la lunghezza del digest n (224, 256, 384 o 512 bit);
- la capacità $c = 2 \cdot n$ del registro interno;
- il bit-rate $r = 1600 - c$, per cui è sempre sufficiente una singola spremitura per ottenere il digest.

La permutazione f opera tra stati di 1600 bit, i quali vengono modellati come parallelepipedi a tre dimensioni, nello specifico di 5×5×64 bit. L'output di f si ottiene applicando per 24 round i seguenti operatori in ordine di apparizione:

1. θ è una mappa lineare che mira alla diffusione dell'informazione all'interno dello stato;
2. χ è composta da traslazioni e ha come obiettivo la dispersione dei dati;
3. π è una permutazione con lo scopo di rompere le periodicità;
4. ρ è un operatore logico e costituisce l'unica trasformazione non lineare;
5. infine, ι rompe le simmetrie tramite l'addizione di valori costanti.

Essendo basata su una costruzione a spugna, SHA-3 può essere anche utilizzata come XOF con un digest di lunghezza n arbitraria. In tal caso, l'algoritmo viene indicato con SHAKE128 o SHAKE256 in base alla capacità scelta ($c/2$).
Al momento i migliori attacchi noti in letteratura per la ricerca di collisioni di SHA-3 avvengono su versioni ridotte a 5 round [29] e in certi casi particolari anche a 6 round [30], non minacciando minimamente la sicurezza della versione completa a 24 round.

5 Confronto tra gli algoritmi

La Tabella 1 riassume le specifiche degli algoritmi di hash descritti nei capitoli precedenti. In particolare, per ogni algoritmo e ogni eventuale versione, vengono confrontate le dimensioni in bit di digest, stato interno e blocchi in cui viene diviso l'input. Viene anche indicato il numero di round all'interno della funzione di compressione per le costruzioni di Merkle-Damgård e della funzione di permutazione per la costruzione a spugna. La colonna "Sicurezza" riporta l'esponente della potenza di 2 nella notazione O-grande del migliore attacco alle collisioni, e si quantifica in bit.
Infine, sono stati raccolti dei benchmark sulle prestazioni per il calcolo del digest di input lunghi 8 byte sul computer "comet" (amd64, Intel Core i3-10110U, 2x2100) risalenti al 30/05/2023 [31], espresse in cicli eseguiti per ogni byte (cpb): più il valore è alto più calcoli saranno necessari e quindi la funzione di hash richiederà più tempo.

Algoritmo	Versione	Digest (bit)	Stato (bit)	Blocchi (bit)	Round	Sicurezza (bit)	Prestazioni (cpb)
MD5	-	128	128	512	64	24	61.12
SHA-1	-	160	160	512	80	64	56.25
SHA-2	SHA-224	224	256	512	64	112	96.00
	SHA-256	256				128	96.00
	SHA-512/224	224	512	1024	80	112	141.00
	SHA-512/256	256				128	141.00
	SHA-384	384				192	135.00
	SHA-512	512				256	141.00
SHA-3	SHA3-224	224	1600	1152	24	112	159.75
	SHA3-256	256		1088		128	158.75
	SHA3-384	384		832		192	163.88
	SHA3-512	512		576		256	163.75
	SHAKE128	n		1344		$d/2$	166.12
	SHAKE256	n		1088		$d/2$	148.50

Tabella 1 - Comparazione tra le funzioni di hash

6 Conclusioni

Per quanto detto nei capitoli precedenti, si raccomanda l'utilizzo solo di funzioni di hash della famiglia di SHA-2 e SHA-3 con le versioni riportate in Tabella 2. Le versioni a 224 bit non sono raccomandate in via precauzionale, in quanto le versioni a 256 bit risultano ad esse comparabili in termini di prestazioni, garantendo un maggior livello di sicurezza. Per quanto riguarda le XOF, sono raccomandate entrambe le versioni di SHAKE. Gli algoritmi MD5 e SHA-1 sono obsoleti e non vengono consigliati per alcuna applicazione.

Algoritmo Raccomandato	Versione
SHA-2	SHA-256
	SHA-512/256
	SHA-384
	SHA-512
SHA-3	SHA3-256
	SHA3-384
	SHA3-512
	SHAKE128
	SHAKE256

Tabella 2 - Funzioni di hash raccomandate

Bibliografia

[1] ACN. *Introduzione alla Crittografia e alle Linee Guida*. Linee Guida Funzioni Crittografiche, 2026. URL: https://www.acn.gov.it/portale/crittografia.

[2] L. K. Grover. «A fast quantum mechanical algorithm for database search». In: *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing (STOC)*. 1996, pp. 212–219. DOI: 10.1145/237814.237866.

[3] A. J. Menezes, P. C. V. Oorschot e S. A. Vanstone. *Handbook of applied cryptography (1st edition)*. CRC Press, 1997. DOI: 10.1201/9780429466335.

[4] D. R. Stinson e M. Paterson. *Cryptography: Theory and Practice (4th edition)*. Chapman e Hall/CRC, 2017. DOI: 10.1201/9781315282497.

[5] ACN. *Firme Digitali*. Linee Guida Funzioni Crittografiche, 2026. URL: https://www.acn.gov.it/portale/crittografia.

[6] ACN e GPDP. *Conservazione delle Password*. Linee Guida Funzioni Crittografiche, 2026. URL: https://www.acn.gov.it/portale/crittografia.

[7] ACN. *Codici di Autenticazione di Messaggi (MAC)*. Linee Guida Funzioni Crittografiche, 2026. URL: https://www.acn.gov.it/portale/crittografia.

[8] R. C. Merkle. «A Certified Digital Signature». In: *Advances in Cryptology - CRYPTO '89*. Lecture Notes in Computer Science. 1990, pp. 218–238. DOI: 10.1007/0-387-34805-0_21.

[9] I. B. Damgård. «A Design Principle for Hash Functions». In: *Advances in Cryptology - CRYPTO '89*. Lecture Notes in Computer Science. 1990, pp. 416–427. DOI: 10.1007/0-387-34805-0_39.

[10] G. Bertoni et al. *Sponge functions*. ECRYPT Hash Functions Workshop. 2007. URL: https://keccak.team/files/SpongeFunctions.pdf.

[11] R. L. Rivest. *The MD5 message-digest algorithm*. RFC 1321. 1992. DOI: 10.17487/RFC1321.

[12] V. Klima. *Tunnels in Hash Functions: MD5 Collisions Within a Minute*. IACR Cryptology ePrint Archive. 2006. URL: https://eprint.iacr.org/2006/105.

[13] A. Sotirov et al. *MD5 considered harmful today, creating a rogue CA certificate*. 25th Annual Chaos Communication Congress. 2008. URL: https://www.win.tue.nl/hashclash/rogue-ca.

[14] M. M. J. Stevens. *On Collisions for MD5*. Eindhoven University of Technology. 2007. URL: https://www.win.tue.nl/hashclash/On%20Collisions%20for%20MD5%20-%20M.M.J.%20Stevens.pdf.

[15] A. A. Kuznetsov. *An algorithm for MD5 single-block collision attack using high-performance computing cluster*. IACR Cryptology ePrint Archive. 2014. URL: https://eprint.iacr.org/2014/871.

[16] NIST. *Secure Hash Function*. FIPS 180-1. U.S. Department of Commerce, 1995. DOI: 10.6028/NIST.FIPS.180-1.

[17] M. Stevens et al. *The first collision for full SHA-1*. IACR Cryptology ePrint Archive. 2017. URL: https://eprint.iacr.org/2017/190.

[18] E. Barker e A. Roginsky. *Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths*. SP 800-131A. NIST, 2011. DOI: 10.6028/NIST.SP.800-131A.

[19] SOG-IS. *Agreed Cryptographic Mechanisms, Version 1.3*. 2023. URL: https://www.sogis.eu/documents/cc/crypto/obsolete/SOGIS-Agreed-Cryptographic-Mechanisms-1.3.pdf.

[20] ANSSI. *Guide des mécanismes cryptographiques, Version 2.04*. 2020. URL: https://cyber.gouv.fr/sites/default/files/2021/03/anssi-guide-mecanismes_crypto-2.04.pdf.

[21] BSI. *Cryptographic Mechanisms: Recommendations and Key Lengths, Version 2023-01*. TR-02102-1. 2023. URL: https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TG02102/BSI-TR-02102-1.

[22] ETSI. *Electronic Signatures and Infrastructures (ESI); Cryptographic Suites, Version 1.4.3*. TS 119 312. 2023. URL: https://www.etsi.org/deliver/etsi_ts/119300_119399/119312/01.04.03_60/ts_119312v010403p.pdf.

[23] A. Chattopadhyay et al. «On the Cost of ASIC Hardware Crackers: A SHA-1 Case Study». In: *Topics in Cryptology - CT-RSA*. Lecture Notes in Computer Science. 2021, pp. 657–681. DOI: 10.1007/978-3-030-75539-3_27.

[24] NIST. *Secure Hash Standard*. FIPS 180-4. U.S. Department of Commerce, 2015. DOI: 10.6028/NIST.FIPS.180-4.

[25] ISO. *IT Security techniques – Hash-functions – Part 3: Dedicated hash-functions*. ISO/IEC 10118-3. 2018. URL: https://www.iso.org/standard/67116.html.

[26] NIST. *Secure Hash Standard*. FIPS 180-2. U.S. Department of Commerce, 2002. URL: https://csrc.nist.gov/pubs/fips/180-2/final.

[27] J. Li, T. Isobe e K. Shibutani. «Converting meet-in-the-middle preimage attack into pseudo collision attack: Application to SHA-2». In: *Fast Software Encryption (FSE)*. Lecture Notes in Computer Science. 2012, pp. 264–286. DOI: 10.1007/978-3-642-34047-5_16.

[28] NIST. *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*. FIPS 202. U.S. Department of Commerce, 2015. DOI: 10.6028/NIST.FIPS.202.

[29] I. Dinur, O. Dunkelman e A. Shamir. «Improved practical attacks on round-reduced Keccak». In: *Journal of cryptology* 27 (2014), pp. 183–209. DOI: 10.1007/s00145-012-9142-5.

[30] J. Guo et al. «Practical collision attacks against round-reduced SHA-3». In: *Journal of Cryptology* 33 (2020), pp. 228–270. DOI: 10.1007/s00145-019-09313-3.

[31] VAMPIRE. *eBACS: ECRYPT Benchmarking of Cryptographic Systems*. [Consultato: Giugno 2023]. URL: https://bench.cr.yp.to/results-hash.html.