UNIVERSITÀ DEGLI STUDI DI ROMA TOR VERGATA

MACROAREA DI SCIENZE MATEMATICHE, FISICHE E NATURALI

CORSO DI LAUREA IN
Informatica

TESI DI LAUREA IN
Algoritmi e Crittografia

TITOLO
Funzioni Hash Crittografiche: studio dei principi progettuali e confronto sperimentale tra Poseidon e SHA-256

Relatore:
Prof.
Francesco Pasquale

Candidato:
Sandu Mihai Alexandru
Matricola: *0327308*

Anno Accademico 2024/2025

"Quanto in grande sogneresti, se sapessi di non poter fallire?"

- Seth Godin

Indice

1 Introduzione

Nel panorama della crittografia moderna, le funzioni hash crittografiche [8, 9] rivestono un ruolo cruciale in numerosi ambiti applicativi, dalla verifica dell'integrità dei dati fino alla sicurezza delle transazioni nelle blockchain. L'importanza di queste primitive si fonda sulla loro capacità di sintetizzare in un *digest* a lunghezza fissa un input di lunghezza arbitraria, garantendo proprietà fondamentali come la *one-wayness*, la *collision resistance* e un comportamento fortemente sensibile a qualsiasi variazione anche minima dell'input. Nel corso del tempo, numerosi algoritmi hash sono stati proposti e adottati, tra cui MD5, SHA-1, SHA-2 e le sue varianti, come SHA-256. Tuttavia, l'avvento di nuovi contesti applicativi – in particolare quelli legati alle prove a conoscenza zero (*Zero-Knowledge Proofs*, ZKP) – ha spinto la ricerca verso nuove direzioni, portando alla nascita di algoritmi come *Poseidon* [3, 4], pensati per essere nativamente efficienti su circuiti aritmetici.

Questa tesi si propone di esplorare le funzioni hash crittografiche, con un focus particolare su Poseidon, confrontandola dal punto di vista teorico e statistico con uno degli algoritmi hash di riferimento più solidi e consolidati, ovvero SHA-256. Il lavoro si articola in diverse fasi: si parte da una trattazione teorica delle proprietà desiderabili per una funzione hash sicura [7, 1] e delle tecniche crittoanalitiche a cui tali funzioni devono resistere [5]. Successivamente, viene analizzato il design interno di Poseidon, ponendo l'accento sui meccanismi strutturali – come la costruzione *sponge*, la rete *Substitution-Permutation* (SPN), la funzione di permutazione Poseidonπ e la matrice *MDS* – che ne determinano sia la sicurezza sia l'efficienza implementativa. L'analisi si completa con un confronto sperimentale tra Poseidon e SHA-256, tramite una batteria di test statistici volti a verificarne le proprietà fondamentali di robustezza.

Poseidon, concepita appositamente per ambienti ZKP, è costruita per minimizzare il numero di vincoli e il grado polinomiale dei circuiti, risultando quindi più adatta rispetto a SHA-256 nei contesti che richiedono efficienza su circuiti a campo finito. A livello di design, Poseidon adotta una struttura di tipo *sponge*, con stato diviso in *rate* e *capacity*, e applica una permutazione interna articolata in round completi e parziali. Tali round includono operazioni di esponenziazione non lineare (*S-box*), moltiplicazione per matrici MDS e l'aggiunta di costanti di round. Questo schema è pensato per ottenere una forte diffusione e confusione, pur mantenendo bassa la complessità aritmetica, ed è supportato da scelte implementative mirate all'efficienza nei circuiti tipici delle blockchain.

Per valutare la solidità di Poseidon sono stati implementati numerosi test statistici, applicati su oltre un milione di input casuali, confrontando i risultati con quelli ottenuti da SHA-256 e, in alcuni casi, anche da MD5. Il primo test affrontato è stato l'*Avalanche Effect*, ovvero la capacità della funzione di reagire in modo imprevedibile a minime variazioni dell'input. I risultati hanno mostrato che Poseidon presenta una

media di cambiamento dei bit del 50.01% con una deviazione standard molto bassa, perfettamente in linea con SHA-256 e superiore a livello di risultati a quella di MD5, confermando la capacità dell'algoritmo di assicurare un output completamente differente anche in presenza di minime modifiche.

Successivamente è stata analizzata la resistenza alle collisioni, sia tramite verifica diretta sull'intero digest sia mediante un'analisi fondata sul *Birthday Paradox*. In entrambi i casi, Poseidon ha mostrato assenza totale di collisioni nei digest completi, e una distribuzione delle collisioni su sottostringhe di n bit coerente con quella teoricamente attesa, mostrando un comportamento indistinguibile da una funzione hash ideale. Il confronto con SHA-256 ha prodotto risultati praticamente sovrapponibili, mentre MD5, pur non evidenziando collisioni nel presente test, resta crittograficamente compromesso in virtù di attacchi noti.

L'analisi dell'uniformità ha permesso di verificare l'equidistribuzione statistica degli output generati. Poseidon ha mostrato un bilanciamento quasi perfetto tra bit a 0 e bit a 1 (rapporto 0.9999) e distribuzioni byte del tutto coerenti con una sorgente casuale uniforme, come confermato anche dal test χ^2, i cui p-value risultano ben al di sopra della soglia di significatività statistica. Ulteriore conferma della qualità crittografica è emersa dal calcolo dell'entropia di Shannon, che per Poseidon si attesta a 7.999993 su un massimo teorico di 8, valore praticamente ideale.

Per esplorare eventuali *bias* o regolarità strutturali, sono stati inoltre eseguiti test di *bit positional analysis* e di autocorrelazione. Anche in questo caso, Poseidon ha dimostrato l'assenza di *bias* significativi su tutte le posizioni dell'output e valori di autocorrelazione compatibili con un comportamento casuale ideale. L'algoritmo ha superato senza anomalie i test di *hash pattern analysis*, con *collision rate* praticamente nullo per finestre di 32 bit e risultati coerenti su finestre più piccole, esattamente come SHA-256.

In sintesi, l'intero lavoro ha evidenziato che Poseidon, pur essendo un algoritmo di concezione recente e ottimizzato per un contesto molto specifico, si comporta in modo solido anche rispetto a criteri classici della crittografia. L'algoritmo ha mostrato un'efficacia statistica paragonabile a SHA-256 su tutti i test considerati, senza evidenziare pattern, *bias* o debolezze strutturali. Tali risultati confermano che Poseidon non è solo una funzione hash efficiente nei protocolli a conoscenza zero, ma rappresenta anche una scelta crittograficamente sicura, potenzialmente utilizzabile in contesti più ampi. La strada rimane comunque aperta a ulteriori approfondimenti crittoanalitici avanzati.

2 Funzioni Hash: Fondamenti Teorici

Le funzioni hash crittografiche sono algoritmi fondamentali che trasformano input di lunghezza arbitraria in output a dimensione fissa (digest), garantendo proprietà di sicurezza come preimage resistance, collision resistance ed effetto valanga. Questo capitolo ne esplora le caratteristiche essenziali, le applicazioni pratiche (dall'integrità dei dati alle blockchain) e gli algoritmi storici (MD5, SHA-256) e moderni (Poseidon). Vengono inoltre analizzati gli attacchi più rilevanti, tra cui crittoanalisi differenziale e lineare, e i metodi per valutarne la robustezza.

2.1 Definizione e proprietà fondamentali

Le funzioni hash crittografiche sono primitive fondamentali in crittografia moderna. Una funzione hash è un algoritmo che riceve in input una stringa di lunghezza arbitraria e produce un output di lunghezza fissa, detto *digest* o *impronta*. Formalmente, si definisce una funzione hash come:

$$H\colon \{0,1\}^* \to \{0,1\}^n$$

dove n è la dimensione dell'output. L'obiettivo primario di una funzione hash crittografica è di agire come una funzione "one-way", ossia facile da calcolare ma difficile da invertire.

Le principali proprietà di sicurezza richieste a una funzione hash crittografica sono:

- **Preimage resistance (One-wayness)**. Dato un valore y nell'immagine della funzione, è computazionalmente difficile trovare un x tale che $H(x) = y$. Questa proprietà garantisce che, conoscendo solo il digest, risulti impraticabile ricostruire l'input originale.
- **Second preimage resistance**. Dato un input x e il suo digest $H(x)$, è difficile trovare un $x' \neq x$ tale che $H(x) = H(x')$.
- **Collision resistance**. È difficile trovare due input distinti x, x' tali che $H(x) = H(x')$.

Oltre a queste tre proprietà classiche, sono desiderabili:

- **Avalanche effect (Effetto valanga)**: Piccole modifiche all'input devono produrre grandi cambiamenti nell'output, in modo che ogni bit dell'output dipenda da ogni bit dell'input in maniera imprevedibile.
- **Uniformità**: I digest prodotti devono essere distribuiti uniformemente nello spazio di output, al fine di evitare bias che possano agevolare attacchi statistici o ridurre l'effettivo spazio delle collisioni.

- **Determinismo**: Per un dato input, la funzione deve sempre produrre lo stesso output, altrimenti risulterebbe inutilizzabile in contesti di verifica.

- **Efficienza computazionale**: La funzione deve essere veloce da calcolare, anche su dispositivi con risorse limitate.

2.2 Applicazioni pratiche

Le funzioni hash crittografiche hanno un'ampia varietà di applicazioni pratiche in sicurezza informatica e sistemi distribuiti:

- **Integrità dei dati**: Sono usate per generare checksum che garantiscono che un file o un messaggio non sia stato modificato.

- **Firme digitali**: Nei protocolli riguardanti le firme, del messaggio da firmare viene prima calcolato il digest e successivamente il digest viene firmato. Questo riduce il carico computazionale e standardizza la lunghezza dei messaggi firmati.

- **Autenticazione dei messaggi**: Utilizzando funzioni hash in combinazione con una chiave segreta si possono costruire dei codici per l'autenticazione dei messaggi, per garantire autenticità e integrità.

- **Password hashing e salting**: Le password degli utenti vengono salvate sotto forma di hash, spesso con aggiunta di un *salt*, per evitare che attacchi non possano risalire alle password in chiaro.

- **Blockchain e Proof-of-Work**: Gli hash sono centrali nel mining di criptovalute: il processo di mining richiede di trovare un input che generi un hash inferiore a un certo target.

- **Zero-Knowledge Proofs**: Alcune funzioni hash sono progettate per essere efficientemente implementabili in circuiti aritmetici, cosi da essere utilizzate nelle prove a conoscenza zero (es. ZK-SNARKs e ZK-STARKs).

2.3 Funzioni Hash Note

Negli ultimi decenni sono stati sviluppati numerosi algoritmi hash crittografici, ciascuno con caratteristiche e livelli di sicurezza differenti. Tra i principali algoritmi che hanno avuto un ruolo significativo nella storia della crittografia troviamo:

- **MD5** (Message Digest 5), progettato da Ronald Rivest nel 1991, produce un output di 128 bit. Basato su un tipo di costruzione detto Merkle-Damgård, MD5 è stato per lungo tempo lo standard per checksum e firma digitale. Tuttavia, dal 2004 sono state scoperte collisioni pratiche (Wang et al.), rendendolo inadeguato per qualunque utilizzo di sicurezza.

- **SHA-256** appartiene alla famiglia SHA-2, standardizzata dal NIST (National Institute of Standards and Technology) nel FIPS 180-4 nel 2001. Produce un digest di 256 bit ed è basata su operazioni bitwise (rotazioni, shift, addizioni modulo 2^{32}) e sulla struttura Merkle-Damgård modificata per resistere ad attacchi noti su SHA-1. SHA-256 è ampiamente utilizzato in:
  - Protocolli di sicurezza (TLS, SSL)
  - Blockchain come Bitcoin per la Proof-of-Work, insieme a RIPEMD-160
  - Funzioni di derivazione di chiavi

  Attualmente non sono note collisioni pratiche, e l'algoritmo è considerato sicuro contro tutte le forme di attacco conosciute, sebbene la sua efficienza nei circuiti a conoscenza zero risulti limitata.
- **RIPEMD** (RACE Integrity Primitives Evaluation Message Digest) è stato progettato nel 1992 dall'EUROCRYPT RACE project, ad oggi viene utilizzata la sua versione migliorata, RIPEMD-160, pubblicata nel 1996, che produce un digest di 160 bit. RIPEMD-160 è stato progettato come alternativa europea a SHA-1, presentando una struttura interna simile ma con funzionamenti che ne aumentano la sicurezza. Ad oggi, RIPEMD-160 rimane considerato sicuro, e viene utilizzato nella Blockchain di Bitcoin.
- **Poseidon** è un algoritmo più recente, progettato nel 2019 da Grassi et al., ottimizzato per circuiti a conoscenza zero. Questo algoritmo verrà poi approfondito nel Capitolo 3.

2.4 Attacchi noti

Benchè le funzioni hash siano costruite per essere resistenti a collisioni e preimage, sono state sviluppate numerose tecniche crittoanalitiche per testarne la robustezza. Tra le principali figurano la crittoanalisi differenziale [2] e la crittoanalisi lineare [6].

2.4.1 Crittoanalisi Lineare

La crittoanalisi lineare è una tecnica nata nell'ambito dei cifrari a blocchi, introdotta da Matsui in [6] per un algoritmo di cirfratura noto come DES, ma applicabile anche alle funzioni hash costruite su cifrari a blocchi. Essa consiste nel trovare *approximate linear relations* tra bit dell'input e dell'output che si verifichino con probabilità significativamente diversa da 0.5. Tali relazioni lineari possono essere sommate e combinate (*linear approximations*) per ridurre lo spazio di ricerca e ottenere informazioni parziali sull'input originale.

Nel contesto delle funzioni hash, la *linear cryptanalysis* è meno efficace rispetto alla differenziale, ma rimane uno strumento teorico importante per valutare la resistenza dell'algoritmo a correlazioni lineari indesiderate.

2.4.2 Crittoanalisi Differenziale

La crittoanalisi differenziale, introdotta da Biham e Shamir in [2], si basa sull'analisi di come differenze specifiche negli input si propagano attraverso le varie operazioni della funzione hash, con l'obiettivo di controllare o predire differenze nell'output.

Questa tecnica è particolarmente efficace sulle funzioni hash costruite su cifrari a blocchi, e ha permesso di sviluppare *collision attacks* pratici. Si studia come una piccola modifica nell'input si propaga attraverso l'algoritmo, calcolando la probabilità che determinate differenze si mantengano dopo ogni step. Identificando differenze con probabilità non trascurabile, è possibile costruire coppie di input che producono lo stesso hash con meno tentativi rispetto a una brute-force.

2.4.3 Altri Attacchi e Verifiche

Oltre agli attacchi crittanalitici, esistono diversi test per valutare la robustezza delle funzioni hash, tra cui:

- Test di Collisione e Birthday test: Verificare la resistenza alla ricerca di due input con lo stesso output, basato sul paradosso del compleanno dove si mostra che per un output di n bit la complessità di trovare collisioni è circa $2^{n/2}$.

- Indipendenza tra bit: Test utilizzato per garantire l'assenza di correlazioni tra le varie posizioni dell'output, verificabile tramite autocorrelazione e calcolo dell'entropia.

- Attacchi Algebrici: come l'Interpolation Attack ricostruisce il cifrario come polinomio per prevedere output. Gröbner Basis Attack che risolve sistemi di equazioni polinomiali per trovare le chiavi. Entrambi sfruttano debolezze algebriche nel design.

3 Poseidon

Poseidon è una funzione hash crittografica progettata per essere efficiente nelle prove a conoscenza zero (ZKP), come SNARKs e STARKs, dove tradizionali algoritmi (es. SHA-256) risultano computazionalmente costosi. Basata su una sponge construction e una permutazione ottimizzata con round completi e parziali, Poseidon minimizza il numero di vincoli e il grado delle equazioni nei circuiti aritmetici, rendendola ideale per applicazioni blockchain e autenticazione privacy-preserving. Questo capitolo ne esplora il design, le componenti chiave (S-box, matrice MDS, costanti di round) e i dettagli implementativi, evidenziando il bilanciamento tra sicurezza ed efficienza.

3.1 Panoramica dell'algoritmo

Negli ultimi anni, l'emergere di sistemi crittografici per la privacy e l'integrità dei dati, in particolare le prove a conoscenza zero (*Zero-Knowledge Proofs*, ZKP), come SNARKs e STARKs, ha richiesto la progettazione di primitive ottimizzate per circuiti aritmetici. Gli Zero-Knowledge Proofs sono tecniche crittografiche che permettono a una parte (**prover**) di dimostrare a un'altra (**verifier**) di conoscere un'informazione segreta senza rivelarla. Devono soddisfare tre proprietà:

- Completezza: Se l'affermazione è vera, il verifier è convinto.
- Soundness (correttezza): Se l'affermazione è falsa, nessun prover malizioso può convincere il verifier.
- Zero-Knowledge: Il verifier non impara nulla oltre alla validità della prova.

In questo contesto, **Poseidon** si è affermata come una funzione hash crittografica progettata specificamente per essere altamente efficiente in questi contesti.

A differenza delle funzioni hash tradizionali, come SHA-256, che risultano computazionalmente costose quando implementate come circuiti aritmetici su campi primi di grande ordine, Poseidon è stata ottimizzata per minimizzare:

- Il numero di vincoli utilizzate
- Il grado delle equazioni polinomiali

Questa caratteristica rende l'algoritmo ideale per tecniche ZKP.

Casi d'Uso Pratici:

- **Autenticazione Senza Rivelare Dati**: Sistemi di accesso passwordless possono usare Poseidon per dimostrare la conoscenza di una credenziale (es. hash di una password) senza trasmetterla.
- **Blockchain e Layer 2**: Poseidon è adottata anche in soluzioni di scaling e privacy su Ethereum e in progetti blockchain che richiedono primitive hash efficienti.

3.2 Caratteristiche Principali e Design

La struttura di Poseidon utilizza una rete di sostituzione-permutazione (SPN) per garantire sicurezza e prestazioni ottimali. La sua costruzione si ispira al framework HADES, che utilizza la stessa strategia per il funzionamento delle permutazioni combinando full round e round parziali.

3.2.1 Costruzione Sponge

Poseidon utilizza la sponge construction, un paradigma generico per costruire funzioni hash su domini di lunghezza arbitraria. La sponge di Poseidon funziona come segue:

- **Stato interno (state)**: di dimensione t, suddiviso in *rate* (r) e *capacity* (c), con $t = r + c$.
- **Assorbimento (Absorbing phase)**: il messaggio in input viene diviso in blocchi di lunghezza r; ciascun blocco viene aggiunto (XOR) con la parte *rate* dello stato, e in seguito viene applicato Poseidonπ, cioè la permutazione interna, su tutto lo stato quindi compresa la capacity. Quest'ultima serve infatti a garantire sicurezza contro gli attacchi in quanto non viene mai esposta direttamente nel digest. Si procede poi con il prossimo blocco
- **Squeezing phase**: una volta assorbito l'intero input, si prende la parte *rate* dello stato come output. Se l'output richiesto è più lungo di r, si applica nuovamente Poseidonπ e si concatena il nuovo blocco *rate* estratto, fino a raggiungere la lunghezza desiderata.

Prendiamo per esempio un stato interno (t) di 5 elementi ($3r$ e 2 c) e sia π la funzione di permutazione e prendiamo in considerazione il messaggio iniziale = [5, 10, 15, 20, 25, 30]. Si parte con la fase di Assorbimento:

$$
\begin{aligned}
&\text{Il nostro stato iniziale sarà} = [0,0,0,0,0] \\
&\quad \text{Blocco 1 del messaggio} = [5,10,15] \\
&\qquad \text{XOR con } \textit{rate} = [0 \oplus 5, 0 \oplus 10, 0 \oplus 15, 0, 0] = [5,10,15,0,0] \\
&\qquad\quad \text{Applica } \pi : [5,10,15,0,0] \xrightarrow{\pi} [8,12,3,7,9] \\
&\quad \text{Blocco 2 del messaggio} = [20,25,30] \\
&\qquad \text{XOR con } \textit{rate} = [8 \oplus 20, 12 \oplus 25, 3 \oplus 30, 7, 9] = [28,21,29,7,9] \\
&\qquad\quad \text{Applica } \pi : [28,21,29,7,9] \xrightarrow{\pi} [4,6,11,13,17]
\end{aligned}
$$

Segue poi la fase di Spremitura dove l'Output estratto è la parte rate = [4, 6, 11]

3.2.2 Permutazione Interna: Poseidonπ

Una **SPN** (Substitution-Permutation Network) è una struttura crittografica che combina operazioni di sostituzione (S-box) e permutazione (P-box) per ottenere confusione e diffusione, essenziali per la sicurezza di un cifrario.

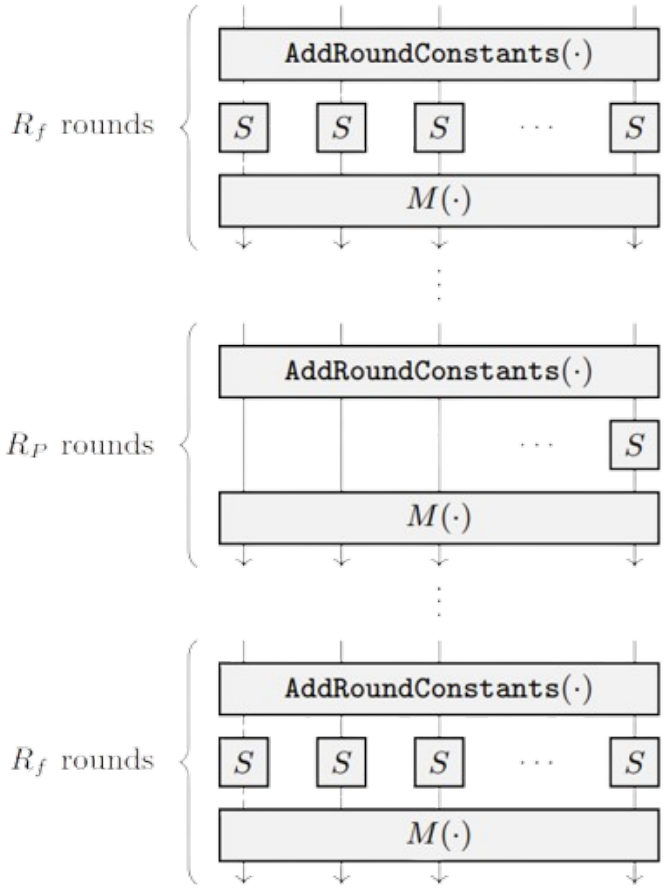


Figura 1: Struttura di HADES

Poseidon utilizza una struttura simile a questa, in particolare la permutazione interna di Poseidon, Poseidonπ, opera su uno stato di dimensione t, applicando ripetutamente round composti da tre passi principali:

1. Addizione di costanti di round (AddRoundConstants)
2. Applicazione dell'S-box
3. Trasformazione lineare MDS (MixLayer)

Ogni round implementa questi step in sequenza. Il numero totale di round varia in base alla dimensione dello stato e al tipo di S-box selezionata, per garantire la sicurezza contro attacchi criptoanalitici noti.

L'S-box è l'unica componente non lineare di Poseidon, in Poseidon si basano su una semplice esponenziazione nel campo finito:

$$S(x) = x^{\alpha}$$

dove α è un esponente scelto per massimizzare la sicurezza e la compatibilità con l'implementazione efficiente. Tipicamente, si raccomandano:

- $\alpha = 5$ per campi primi
- $\alpha = 3$ in altri casi

L'esponenziazione è invertibile e fornisce sufficiente non linearità contro attacchi lineari e differenziali.

Poseidon come detto è progettato con una combinazione di round, in particolare:

- **Full rounds**, in cui tutte le celle dello stato subiscono l'S-box.
- **Partial rounds**, in cui solo la prima cella subisce l'S-box mentre il resto dello stato subisce la sola trasformazione lineare.

Utilizzare round parziali riduce il costo computazionale totale mantenendo comunque adeguata la diffusione della non linearità. Oltre a ciò si utilizza una trasformazione lineare MDS (Maximum Distance Separable) per garantire una rapida diffusione dell'informazione su tutto lo stato.

Definizione: Una matrice MDS di dimensione $t \times t$ è una matrice con proprietà tali per cui ogni sottoinsieme di t colonne è linearmente indipendente.
Scopo: Propagare ogni cambiamento su una cella dello stato a tutte le altre celle in un singolo passo di MixLayer, massimizzando la diffusione e proteggendo contro attacchi di tipo differenziali o strutturali.

Si scelgono matrici MDS con struttura semplice in modo da non andare ad incidere in modo significativo sulle prestazioni. Una costruzione comune è:

$$M_{i,j} = c + d \cdot \delta_{i,j}$$

dove c, d sono due costanti e $\delta_{i,j}$ è la funzione delta di Kronecker.

Le **costanti di round** (da non confondere con le costanti usate nella MDS), vengono aggiunte ad ogni cella dello stato all'inizio di ogni round per garantire:

- L'assenza di simmetrie che potrebbero portare ad attacchi strutturali.
- La rottura di potenziali pattern in input fissi o nulli.
- Una randomizzazione aggiuntiva che garantisce l'unicità di ogni round.

Queste costanti vengono generate tramite un PRNG (generatore numeri pseudocasuali in base ad un seed) o da una funzione di derivazione delle chiavi, ma sono fissate e note pubblicamente.

3.2.3 Composizione dei Round

La permutazione completa di Poseidon si costruisce con:

- R_F full rounds iniziali
- R_P partial rounds centrali
- R_F full rounds finali

Questa struttura "sandwich" (full–partial–full) assicura:
Nei primi full round si assicura un mixing completo di tutte le celle dello stato, successivamente si opera solamente sulla singola cella in modo tale da ridurre la complessità computazionale senza intaccare però la sicurezza. Infine viene fatto un ulteriore mixing finale per garantire che ogni cella dipenda da tutte le altre in modo altamente non lineare.

3.2.4 Parametri di Design

I parametri di Poseidon, come la MDS o il numero di round, vengono scelti sulla base di: Dimensione del campo, livello di sicurezza target, dimensione del digest (tipicamente 128 o 256 bit) ed efficienza nell'implementazione.

Gli autori forniscono parametri consigliati per ogni combinazione di campo e sicurezza target nel paper originale.

3.3 Dettagli Implementativi

L'implementazione di Poseidon richiede un'attenta gestione delle operazioni aritmetiche, della struttura dei round e dell'integrazione con la costruzione sponge. Questa opera su un campo finito primo $\mathbb{F}_p$, dove p è un numero primo vicino a 2^{256}. Nella nostra implementazione, abbiamo utilizzato:

$$p = 2^{256} - 2^{32} - 977$$

Lo stesso valore impiegato in Ethereum per gli indirizzi e le firme digitali.

Tale scelta garantisce compatibilità con le principali piattaforme blockchain e consente una rappresentazione efficiente su 256 bit, oltre a offrire una buona sicurezza contro attacchi algebrici.

Le operazioni modulari sono implementate tramite la libreria `gmpy2`, necessaria per gestire somma, prodotto ed elevamenti a potenza in modo rapido ed efficiente. In particolare, l'S-box applica un'esponenziazione con $\alpha = 5$, scelta implementativa che bilancia velocità e resistenza crittanalitica, come mostrato nella funzione specifica in appendice.

Le costanti di round sono generate in modo deterministico a partire da un seed fisso usando un generatore di numeri casuali e ridotte poi a modulo p.

La matrice MDS è costruita come matrice di Cauchy, che garantisce massima diffusione possibile, inoltre la sua struttura permette ottimizzazioni nella moltiplicazioni con vettori, riducendo il numero di operazioni necessarie rispetto ad altre matrici MDS.

Calcolo degli elementi:

- **Diagonali**: $M_{i,i} = \sum_{k \neq i} \frac{x_i}{x_i - x_k} \mod p$
- **Altri elementi**: $M_{i,j} = \frac{1}{x_i - x_j} \mod p$

La struttura dei round prevede l'alternanza di round completi e parziali. I round completi, otto in totale, applicano l'S-box a tutti gli elementi dello stato e includono l'addizione delle costanti di round e la moltiplicazione per la matrice MDS. I round parziali, invece, sono 56 e applicano l'S-box solo al primo elemento, migliorando l'efficienza complessiva.

L'implementazione è quindi stata sviluppata seguendo le specifiche originali di Poseidon, garantendo un buon equilibrio tra sicurezza, efficienza e adattabilità a contesti reali. I dettagli completi delle funzioni utilizzate sono riportati in appendice per consentire la riproduzione dei risultati.

4 Metodologia di Analsi e Test Statistici

Il presente lavoro non ha come obiettivo la valutazione delle prestazioni computazionali o l'efficienza implementativa di tali algoritmi, bensì si concentra sulla robustezza crittografica dell'algoritmo Poseidon nei confronti di test statistici e probabilistici per la verifica di debolezze.

Per eseguire questi test sono stati generati casualmente 1.000.000 di file da $1KB$ ciascuno e processati dalle funzioni hash Poseidon, Sha256 e MD5. I risultati sono stati registrati poi per un confronto statistico.

Tutto il codice utilizzato per eseguire i test descritti in questo capitolo è pubblicamente disponibile nel mio repository github (https://github.com/alesandu).

4.1 Avalanche Effect

L'**Avalanche Effect** (effetto valanga) è una proprietà fondamentale delle funzioni hash crittografiche che garantisce come un piccolo cambiamento nell'input generi un output completamente diverso. In altre parole, modificare anche un singolo bit nell'input dovrebbe produrre un hash in cui circa il 50% dei bit è diverso rispetto all'hash originale.

Questa proprietà è cruciale perché:

- Impedisce di dedurre informazioni sull'input osservando l'output, diventa qiundi più difficile trovare preimmagini.
- Rende difficile trovare collisioni o invertire la funzione hash.

Per misurare l'Avalanche Effect, si seguono i seguenti passi:

1. **Generazione di input**: Si parte da un input casuale (es. una stringa o un file) e si genera un secondo input modificando **un singolo bit** (es. invertendo il bit nella posizione k).
2. **Calcolo degli hash**: Si calcolano gli hash dei due input:
   $H_1 = \text{Hash}(M_1)$
   $H_2 = \text{Hash}(M_2)$
   Dove M è il messaggio dato in input
3. **Confronto bit a bit**: Si confrontano H_1 e H_2 bit a bit utilizzando l'**operazione XOR** ($\oplus$).
   Differenze $= H_1 \oplus H_2$
4. **Risultato**: Il risultato è una sequenza di bit in cui:
   - **1** indica che il bit è cambiato.
   - **0** indica che il bit è rimasto uguale.

5. **Conteggio** dei bit modificati: Si calcola la **Hamming Distance**, cioè il numero di bit diversi tra H_1 e H_2:

$$\text{Hamming Distance} = \sum_{i=1}^{n} (H_1[i] \oplus H_2[i]) \tag{1}$$

   dove n è la lunghezza in bit dell'hash.

6. **Calcolo della percentuale di cambiamento**: Si determina la frazione di bit modificati:
   Percentuale cambiamento $= \left(\frac{\text{Hamming Distance}}{n}\right) \times 100$

7. **Ripetizione e media**: L'esperimento viene ripetuto su migliaia di coppie di input per ottenere una stima statistica robusta. Si calcola la media delle percentuali di cambiamento e si verifica se si avvicina al 50%.

- Se la media è **vicina** al 50%: La funzione hash soddisfa bene l'Avalanche Effect.
- Se la media è **significativamente diversa** dal 50%: La funzione hash potrebbe avere debolezze crittografiche.

È importante in questi casi calcolare anche **Deviazione standard** in quanto una bassa deviazione standard indica che l'effetto è consistente per input diversi.

Supponiamo di avere:

- Input originale: `"Hello"` → Hash: `00101101...` (256 bit)
- Input modificato (cambiato un bit): `"Helli"` → Hash: `11010010...` (256 bit)

Confronto XOR:

```
H1: 00101101...
H2: 11010010...
XOR:11111111... (supponiamo che tutti i bit siano cambiati)
```

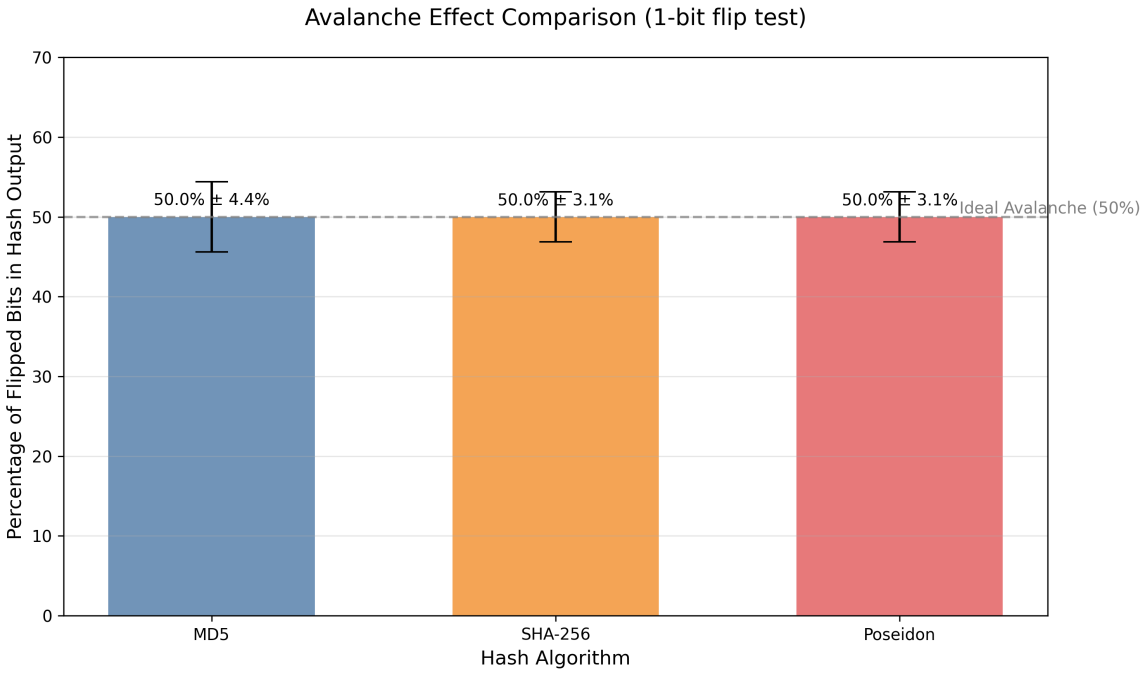
In questo caso, la Hamming Distance è 256 (tutti i bit diversi), quindi la percentuale di cambiamento è $(256/256) \times 100 = 100\%$.
Tuttavia, in un caso reale, ci aspettiamo circa **128 bit cambiati** (50%) in media.

4.2 Risultati Avalanche

In questa sezione vengono riportati i risultati dell'**Avalanche Effect Test**, che misura la sensibilità degli algoritmi hash a variazioni minime dell'input. In particolare, è stato eseguito un test di flip su un singolo bit nel primo byte dell'input, calcolando la differenza media di bit nell'output hash risultante.

I risultati ottenuti sono i seguenti:

- **MD5**: la differenza media di bit è pari al **50.01%** con una deviazione standard di **4.41%**. Questo indica che, in media, l'algoritmo produce una variazione su metà dei bit dell'output, come previsto dalla proprietà di effetto valanga, ma con una deviazione standard leggermente superiore a SHA-256 e Poseidon.
- **SHA-256**: ha mostrato una differenza media di **50.00%** con una deviazione standard di **3.13%**, evidenziando un comportamento estremamente stabile e coerente con la teoria. La bassa deviazione standard conferma che l'effetto valanga è consistente su differenti input testati.
- **Poseidon**: ha prodotto una differenza media di **50.01%** con una deviazione standard di **3.12%**, risultando comparabile a SHA-256 sia come valore medio sia come stabilità. Ciò evidenzia che l'algoritmo Poseidon garantisce un effetto valanga ottimale, come richiesto per una funzione hash crittograficamente sicura.



In conclusione, tutti e tre gli algoritmi analizzati presentano una **percentuale di cambiamento dei bit vicina al valore teorico ideale del 50%**, confermando l'elevata sensibilità agli input e l'assenza di correlazioni lineari prevedibili. Inoltre, le basse deviazioni standard registrate, in particolare per SHA-256 e Poseidon, suggeriscono che l'effetto valanga sia applicato in modo uniforme su un ampio insieme di input, garantendo robustezza contro attacchi che si basano su analisi di propagazione delle differenze.

4.3 Collision Resistance e Birthday Paradox

La **Collision Resistance**, o resistenza alle collisioni, è una proprietà fondamentale delle funzioni hash crittografiche che garantisce l'impraticabilità computazionale di trovare due input distinti che producano lo stesso digest. Formalmente, una funzione hash H è collision-resistant se risulta difficile trovare $x \neq x'$ tali che $H(x) = H(x')$.

Questa proprietà è cruciale perché:

- Impedisce la sostituzione fraudolenta di un messaggio con un altro avente lo stesso hash (es. nella firma digitale, dove un attaccante potrebbe firmare un messaggio benigno e riutilizzare la firma su un messaggio malevolo con hash uguale).
- Costituisce la base di sicurezza per protocolli crittografici, blockchain, firme digitali e certificati SSL.

4.3.1 Birthday Paradox

Il **Paradosso del Compleanno** è un risultato probabilistico che evidenzia come la probabilità di collisione tra valori generati casualmente cresca rapidamente all'aumentare del numero di campioni. Più precisamente, per una funzione hash con output di n bit, la probabilità di trovare almeno una collisione dopo aver calcolato k hash distinti è approssimata da:

$$P \approx 1 - e^{-\frac{k^2}{2^{n+1}}} \tag{2}$$

Quando $k \approx 2^{n/2}$, la probabilità di collisione supera il 50%. Ad esempio, per SHA-256 così come per Poseidon, che producono output di 256 bit, servono circa 2^{128} input distinti per avere il 50% di probabilità di trovare una collisione (bound computazionalmente irraggiungibile con le risorse attuali).

Metodologia di Test Per analizzare la collision resistance di una funzione hash in modo empirico, si segue la procedura:

1. **Generazione input casuali**
   Si generano un numero elevato di input distinti (file, stringhe o numeri casuali) al fine di testare l'algoritmo su un ampio dominio.
2. **Calcolo degli hash**
   Si calcola $H(x_i)$ per ciascun input x_i e si memorizzano i digest ottenuti.
3. **Verifica collisioni**
   Si controlla l'esistenza di coppie (x_i, x_j) con $i \neq j$ tali che $H(x_i) = H(x_j)$.

4. **Calcolo del tasso di collisione osservato**
   Si determina il numero totale di collisioni trovate e si confronta con la probabilità teorica prevista dal paradosso del compleanno. Ciò che si può fare è quindi prendere i primi n bit dei vari digest e controllare se utilizzando k campioni si verifichino collisioni

In base ai risultati possiamo avere:

- **Assenza di collisioni**
  Se non vengono rilevate collisioni anche con un alto numero di input, la funzione si comporta coerentemente con un hash ideale.

- **Collisioni rilevate**
  La presenza di collisioni con un numero di input significativamente inferiore a $2^{n/2}$ evidenzia una potenziale vulnerabilità (come accaduto storicamente per MD5 e SHA-1).

4.4 Risultati Collision Test

In questa sezione vengono riportati i risultati dell'analisi di collision resistance condotta sulle tre funzioni hash considerate. Come spiegato nella metodologia, il test ha previsto due fasi: la verifica delle collisioni sull'intero output degli algoritmi e l'analisi del **Birthday Paradox** sulle sottostringhe di n bit.

La verifica delle collisioni sui digest completi (128 bit per MD5, 256 bit per SHA-256 e Poseidon) ha prodotto i seguenti risultati:

- **MD5**: 0 collisioni
- **SHA-256**: 0 collisioni
- **Poseidon**: 0 collisioni

Questi risultati confermano che, per il campione di input testato, non sono state osservate collisioni sull'intero output, come previsto dall'elevata dimensione dello spazio di codominio delle funzioni hash analizzate. Va tuttavia ricordato che **MD5 è stato storicamente rotto proprio attraverso attacchi di collisione pratici**, motivo per cui non è stato incluso nella seconda fase dell'analisi relativa al Birthday Paradox.

Poiché il Birthday paradox ci indica che output di lunghezza pari a 256bit sono al momento impossibili da testare, ciò che si può fare è prendere sottostringhe di n bit dei vari digest generati e testare se ci sono collisioni tra questi ultimi. Infatti la seconda parte dell'analisi si concentra sul confronto della frequenza di collisioni per Poseidon e SHA-256 considerando i primi **n bit** dei rispettivi output, in modo da verificare la coerenza empirica con il paradosso del compleanno. La tabella seguente riporta i risultati per i valori di n selezionati:

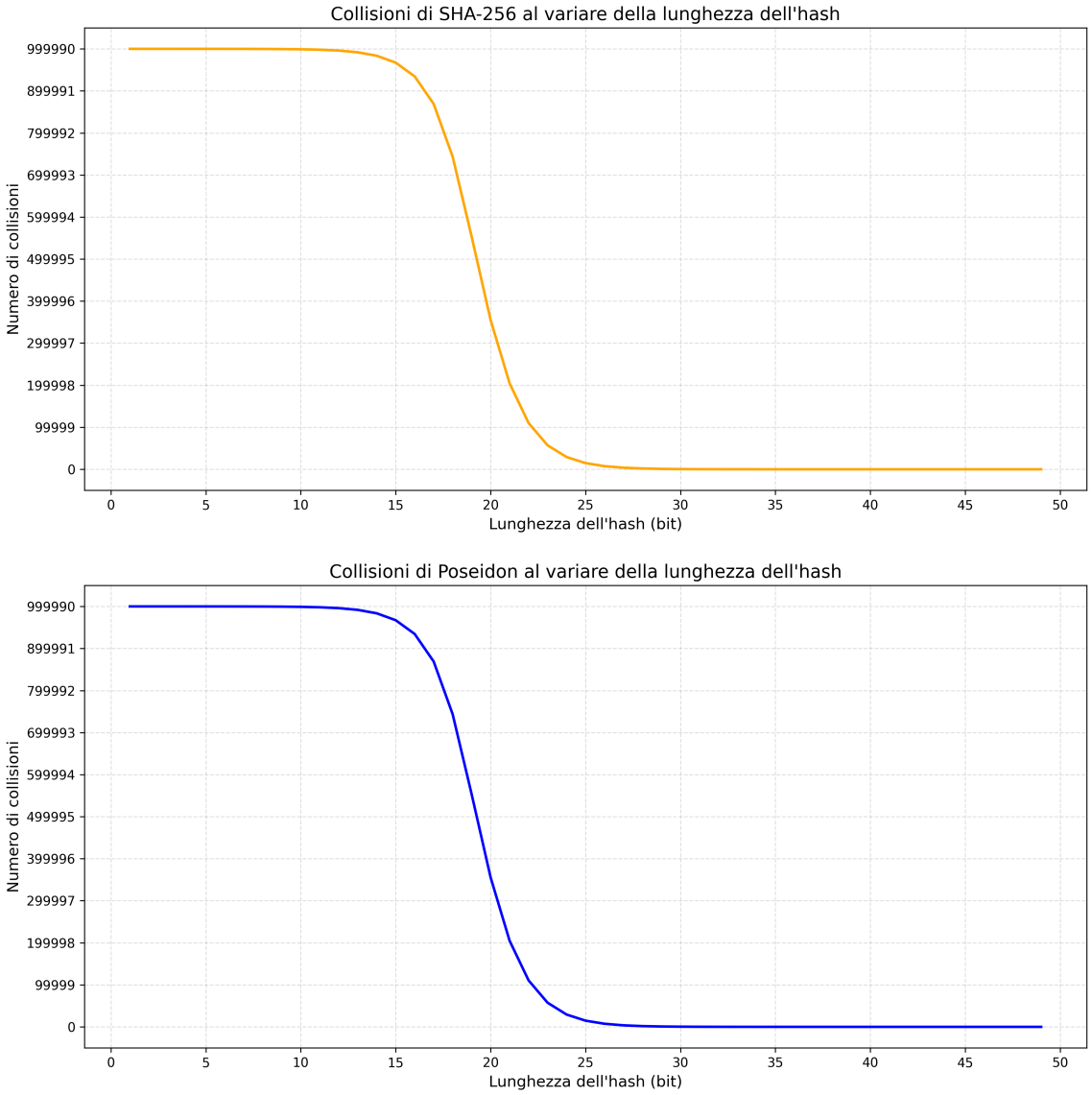


Figura 2: Confronto della distribuzione delle collisioni tra SHA-256 e Poseidon

Bits (n)	Collisioni Poseidon	Collisioni SHA-256
10	998976	998976
15	967232	967232
20	355384	355017
25	14728	14717
30	424	454
35	18	12
40	1	0
45	0	0
50	0	0

Per valori di n inferiori a 20, si osserva una frequenza significativamente elevata di collisioni. Tale comportamento è coerente con le attese teoriche: poiché sono stati generati 1.000.000 di digest, la soglia critica per la quale la probabilità di collisioni diventa non trascurabile è approssimativamente $\log_2(1.000.000) \approx 19.93$ bit.

Finché n rimane al di sotto di questo valore, lo spazio degli hash risulta insufficiente a garantire l'unicità dei digest con alta probabilità, determinando inevitabilmente la presenza di duplicazioni. Per n inferiori a 32, la probabilità di trovare collisioni è ancora massima, Per n superiori a 32, invece, si registra una rapida riduzione delle collisioni, tendente a zero per n $\geq$ 43.

L'analisi conferma quindi che:

- Poseidon e SHA-256 si comportano come **funzioni hash ideali**, mostrando collisioni in linea con la distribuzione probabilistica attesa al variare di n.

- MD5, sebbene in questo test non abbia mostrato collisioni sull'intero output, **non può essere considerato sicuro** in virtù degli attacchi pratici già esistenti che ne compromettono la collision resistance.

4.5 Uniformità e Chi-square Test

L'**uniformità** è una proprietà fondamentale delle funzioni hash crittografiche che garantisce che l'output prodotto sia distribuito in modo equo su tutto lo spazio di codominio. In un hash ideale, ogni bit ha la stessa probabilità di essere 0 o 1 (pari al 50%) e ogni possibile byte ha la stessa probabilità di occorrenza, assicurando l'assenza di bias che potrebbero essere sfruttati per ridurre la complessità degli attacchi.

Una distribuzione non uniforme potrebbe indicare:

- **Bias statistici**: alcuni valori risultano più probabili di altri, riducendo l'effettiva entropia e quindi la sicurezza.

- **Debolezze crittografiche**: possibili correlazioni tra input e output o strutture prevedibili nell'hash.

Metodologia del Test Per verificare empiricamente l'uniformità di una funzione hash si segue questa procedura:

1. **Generazione di input casuali**
   Si creano migliaia di input distinti (es. stringhe o file randomici).

2. **Calcolo degli hash**
   Si calcola l'hash di ciascun input, ottenendo un insieme di digest da analizzare.

3. **Conteggio delle frequenze**
   Si contano le occorrenze di ciascun byte e bit nell'intero insieme di output generati.

4. **Calcolo delle probabilità empiriche**

   - Per ogni byte i, si calcola la probabilità empirica come:

   $$P(i) = \frac{\text{occorrenze}(i)}{\text{numero totale di byte analizzati}} \tag{3}$$

   - Per i bit invece bastarà contare le occorrenze di 0 e 1 ottenute.

5. **Valutazione dell'uniformità**

   - In un hash ideale, $P(i) \approx \frac{1}{256}$ per ogni byte i. Deviazioni significative possono indicare un bias.
   - In un hash ideale la probabilità che un certo bit sia 0 o 1 dovrà essere equiprobabile, di conseguenza la quantità di 0 e 1 nel digest dovrà essere a sua volta ≈ 0.5.

4.5.1 Chi-square

Per quantificare rigorosamente l'uniformità si utilizza il **test chi-quadro (χ^2)**, che confronta la distribuzione empirica osservata con la distribuzione uniforme teorica attesa. La statistica χ^2 è calcolata come:

$$\chi^2 = \sum_{i=0}^{255} \frac{(O_i - E)^2}{E} \tag{4}$$

dove:

- O_i è il numero di occorrenze osservate del byte i,
- E è il numero di occorrenze attese per ciascun byte, calcolato come:

$$E = \frac{\text{numero totale di byte analizzati}}{256} \tag{5}$$

Una volta ottenuto il valore χ^2 lo si confronterà con le tabelle di chi-quadro che forniscono i valori critici; Nel nostro caso il grado di libertà *df* sarà pari a 256 (numero di byte possibili) e la probabilità attesa sarà di 0.39%.

χ^2 **vicino al valore teorico atteso** (per il grado di libertà considerato): indica che la distribuzione osservata non differisce significativamente da quella uniforme, confermando una buona uniformità e imprevedibilità dell'hash.

χ^2 **elevato rispetto al valore critico**: suggerisce che alcuni byte si presentano con frequenza maggiore o minore rispetto all'atteso, evidenziando potenziali bias nell'algoritmo.

L'analisi combinata di **uniformità, chi-square** consente di verificare che l'output di una funzione hash sia equidistribuito, garantendo la resistenza a possibili attacchi statistici, come quelli usati nella crittoanalisi differenziale e confermando che l'hash sia imprevedibile e ben distribuito.

4.6 Risultati Per Uniformità e Chi-square

In questa sezione si riportano i risultati dei test di uniformità e chi-quadro eseguiti sulle tre funzioni hash considerate: MD5, SHA-256 e Poseidon. Come descritto nella metodologia, l'obiettivo di tali analisi è verificare l'assenza di bias significativi nella distribuzione dei bit e dei byte generati dagli algoritmi, garantendo che l'output sia indistinguibile da una sequenza casuale.

Per quanto riguarda la distribuzione dei bit (0 e 1), i risultati mostrano:

- **MD5**: sono stati contati 63.995.769 bit a 0 e 64.004.231 bit a 1, con un rapporto complessivo pari a 0,9998, molto prossimo al valore ideale di 1. Ciò indica un'eccellente distribuzione equiprobabile dei bit.
- **SHA-256**: sono stati contati 128.022.132 bit a 0 e 127.977.868 bit a 1, con un rapporto pari a $1,0003$. Anche in questo caso, la funzione mostra una distribuzione pressoché perfetta tra 0 e 1.
- **Poseidon**: ha registrato 127.999.589 bit a 0 e 128.000.411 bit a 1, con un rapporto di $0,9999$, confermando un'ottima uniformità bitwise.

Tutti e tre gli algoritmi hanno mostrato **nessuna deviazione significativa dalla distribuzione uniforme**, risultando coerenti con la progettazione di funzioni hash crittografiche sicure.

Per quel che invece riguarda il conteggio di byte, abbiamo che per le frequenze dei primi byte più comuni si ottengono i seguenti risultati:

Questi conteggi evidenziano come le occorrenze dei singoli byte si distribuiscano in modo equilibrato su tutto il dominio di output, senza picchi anomali che possano suggerire bias significativi.

MD5 [62566, 62614, 62155, 62760, 62891, 62377, 62534, ...]
SHA-256 [124640, 124876, 125651, 124786, 125637, 125135, 125027, ...]
Poseidon [124285, 125365, 124764, 124909, 125228, 125082, 125434, ...]

MD5 [0,391%, 0,391%, 0,388%, 0,392%, 0,393%, 0,389%, 0,390%, ...]
SHA-256 [0,389%, 0,390%, 0,392%, 0,389%, 0,392%, 0,391%, 0,390%, ...]
Poseidon [0,388%, 0,391%, 0,389%, 0,390%, 0,391%, 0,390%, 0,391%, ...]

Poiché i byte per essere equiprobabili devono avere una probabilità pari a $1/256 \approx 0.00391$ le probabilità empiriche calcolate confermano i risultati attesi dalla teoria.

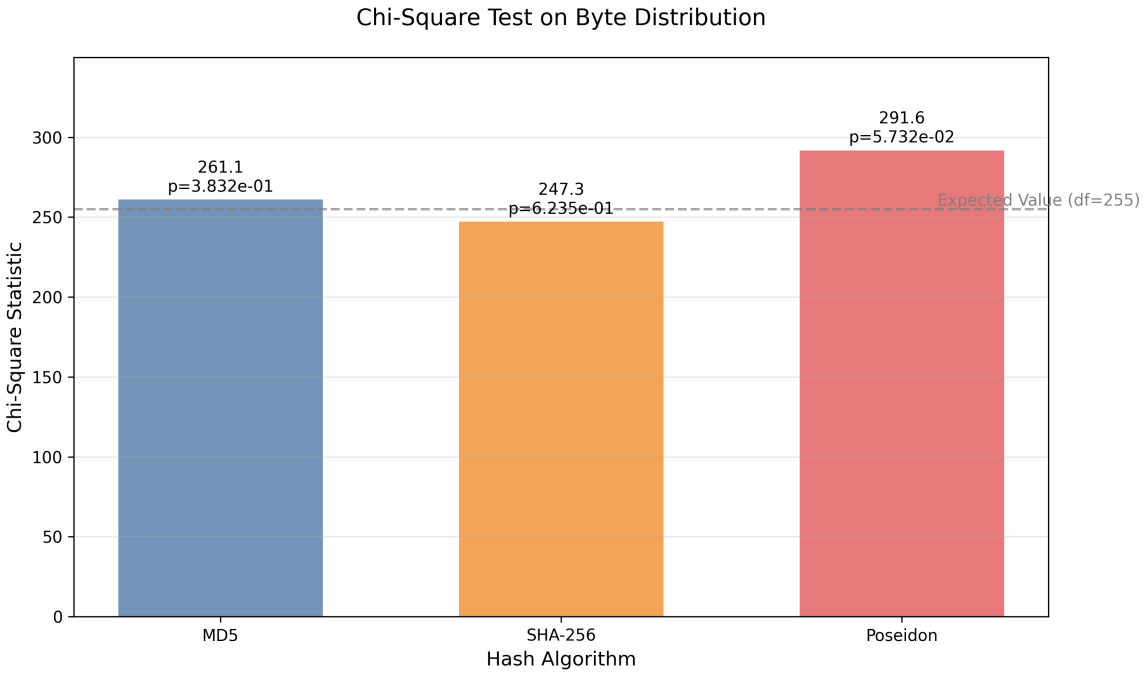
L'analisi del test Chi-Quadro, invece, ha prodotto i seguenti risultati:

Algoritmo	χ^2	*p-value*
MD5	261,089248	0,3832
SHA-256	247,3044	0,6234
Poseidon	291,585648	0,0573

Il p-value nel test chi-square è una misura statistica che aiuta a determinare la significatività dei risultati di un test di ipotesi. Un p-value alto (>0.05) significa che non ci sono prove per rifiutare l'ipotesi.

Un p-value basso (<0.05 o <0.01) invece significa che i risultati differiscono significativamente dall'uniformità. (i byte non sono distribuiti in modo uniforme come ci si aspetterebbe).

I valori di questi sono tutti superiori a 0,05 per MD5 e SHA-256 e Poseidon evidenziando quindi come non esistano evidenze statisticamente significative per rifiutare l'ipotesi nulla di distribuzione uniforme. Tali valori infatti confermano che tutti gli algoritmi presentano una distribuzione dei byte compatibile con la distribuzione uniforme teorica attesa.



In conclusione, l'analisi dell'uniformità bitwise, delle distribuzioni byte e del test chi-quadro evidenzia come **MD5, SHA-256 e Poseidon garantiscano una distribuzione adeguatamente uniforme dei propri output**, requisito essenziale per la sicurezza crittografica, l'assenza di bias sfruttabili e la corretta implementazione degli algoritmi.

4.7 Bit Positional Analysis e Shannon Entropy

L'analisi combinata della **Bit Positional Analysis** e dell'**Shannon Entropy** fornisce un quadro approfondito sulla distribuzione e casualità dei bit nell'output di una funzione hash, aspetti fondamentali per garantire la sicurezza crittografica.

La **Bit Positional Analysis** valuta la distribuzione di 0 e 1 per ciascuna posizione di bit all'interno degli hash generati. Un hash ideale deve garantire che ogni bit abbia probabilità pari al 50% di essere 0 o 1, indipendentemente dalla sua posizione.

Metodologia

1. **Generazione input casuali**
   Si generano un numero elevato di input distinti (file, stringhe o numeri casuali) al fine di testare l'algoritmo su un ampio dominio.

2. **Calcolo degli hash**
   Si calcola $H(x_i)$ per ciascun input x_i e si memorizzano i digest ottenuti.

3. **Conteggio delle frequenze per bit**
   Per ogni posizione i (dove $i = 1, 2, ..., n$ con n la lunghezza in bit dell'hash), si contano:
   - Il numero di volte in cui il bit è 0 (C_i^0)
   - Il numero di volte in cui il bit è 1 (C_i^1)
4. **Calcolo delle probabilità empiriche**

$$P_i^0 = \frac{C_i^0}{N}, \quad P_i^1 = \frac{C_i^1}{N}$$

   dove N è il numero totale di hash analizzati.

Una volta calcolati P_i^0 e P_i^1 si analizza il valore. Se le probabilità sono **vicine** a 0.5 allora si ha un'ottima distribuzione e assenza di bias posizionali, altrimenti **deviazioni significative** da 0.5 possono evidenziare bias specifici, riducendo l'imprevedibilità dell'hash e introducendo possibili debolezze sfruttabili tramite attacchi differenziali.

4.7.1 Shannon Entropy

La **Shannon Entropy** quantifica l'incertezza media associata a ogni byte dell'hash, rivelando eventuali bias non catturati dai conteggi semplici. Per calcolare quest'ultima si utilizza la seguente formula:

Definizione: Data una distribuzione di probabilità $P(i)$ per ciascun byte i (dove $i = 0, 1, ..., 255$), l'entropia è calcolata come:

$$H = -\sum_{i=0}^{255} P(i) \cdot \log_2 P(i)$$

Una volta generati i digesti si conta il numero di occorrenze per ciascun byte, e si calcola la probabilità $P(i)$:

$$P(i) = \frac{\text{occorrenze}(i)}{\text{numero totale di byte analizzati}}$$

L'Entropia massima teorica avrà un valore pari a **8 bit per byte**, corrispondente a una distribuzione uniforme su tutti i possibili valori byte (256 valori equiprobabili) ($\log_2(256) = 8$).

Di conseguenza, un'Entropia alta ($H \geq 7.99$) indica che l'hash è statisticamente imprevedibile, senza pattern sfruttabili.

Al contrario, un'Entropia bassa ($H < 7.95$) indica la presenza di ridondanze o bias nella distribuzione dei byte, riducendo la sicurezza dell'algoritmo utilizzato contro attacchi statistici.

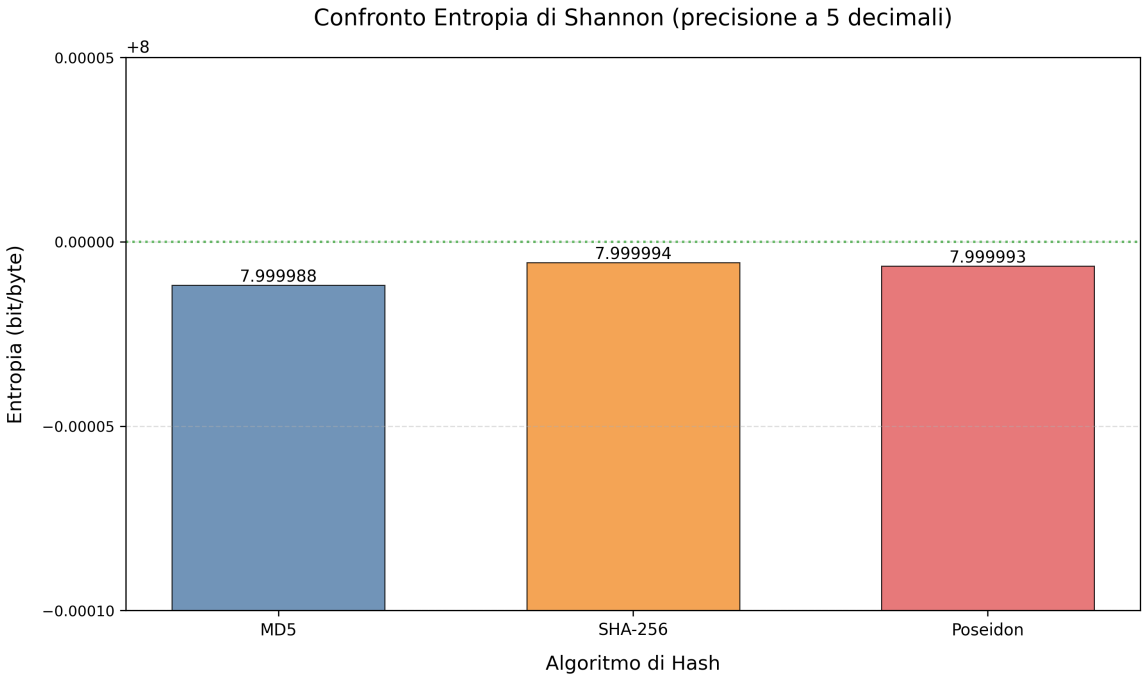
Queste analisi, confermano se l'algoritmo hash implementa una distribuzione pseudo-casuale sufficientemente uniforme e priva di bias, garantendo robustezza contro attacchi crittografici avanzati.

4.8 Risultati per Bit Position Analysis e Shannon Entropy

In questa sezione vengono presentati i risultati dell'analisi combinata di **Bit Positional Analysis** e **Shannon Entropy**, eseguite per verificare l'assenza di bias posizionali e la massima casualità statistica degli output generati dagli algoritmi hash.

I valori di entropia di Shannon calcolati per ciascun algoritmo sono i seguenti:

- **MD5**: `7.99998`
- **SHA-256**: `7.999994`
- **Poseidon**: `7.999993`



Tutti e tre gli algoritmi hanno prodotto valori estremamente prossimi all'entropia massima teorica di **8.0 bit per byte**, confermando un'elevata imprevedibilità e l'assenza di pattern ricorrenti nei digest generati. In particolare, SHA-256 e Poseidon hanno mostrato valori praticamente indistinguibili dall'entropia ideale, mentre MD5 ha registrato un valore leggermente inferiore ma comunque compatibile con le fluttuazioni statistiche attese.

L'analisi della distribuzione dei bit in ciascuna posizione dell'output (da 0 a 127 per MD5, da 0 a 255 per SHA-256 e Poseidon) non ha evidenziato alcun bias significativo:

- **MD5**: 0 posizioni con bias superiore allo 0.15%
- **SHA-256**: 0 posizioni con bias superiore allo 0.15%
- **Poseidon**: 0 posizioni con bias superiore allo 0.15%

Questi risultati indicano che per nessuna delle funzioni hash analizzate sono state rilevate posizioni con deviazioni significative rispetto alla distribuzione ideale (La probabilità che un bit in posizione x sia 0 (o 1) è quindi compresa tra 50.15% e 49.85%).

Di conseguenza, gli algoritmi garantiscono una distribuzione **uniforme e indipendente su tutti i bit** dell'output, requisito fondamentale per assicurare la robustezza contro attacchi statistici o differenziali.

Con una soglia più stringente (threshold di 0.10), emergono piccole deviazioni:

- **MD5**
  - 6 bit con bias (su 128), con deviazioni minime (max: $\pm 0.17\%$).
  - Esempio: Bit 29 (49.84% vs 50.17%).

  Analisi: Le deviazioni sono marginali e potrebbero essere dovute a limitazioni del campione analizzato o a caratteristiche intrinseche dell'algoritmo. MD5, sebbene obsoleto per usi crittografici, mostra comunque un comportamento quasi uniforme.

- **SHA-256**
  - 7 bit con bias (su 256), con deviazioni $\leq \pm 0.15\%$.
  - Esempio: Bit 42 (49.85% vs 50.15%).

  Analisi: SHA-256, progettato per alta sicurezza, conferma la sua robustezza. Le deviazioni osservate sono trascurabili e non indicative di vulnerabilità.

- **Poseidon**
  - 17 bit con bias (su 256), con alcune deviazioni più marcate (es. Bit 104: 50.16% vs 49.84%).

  Analisi: Essendo un hash progettato per ambienti specifici (es. ZKP), potrebbe mostrare più variabilità in alcuni bit, ma le deviazioni restano entro margini accettabili per la maggior parte degli use case.

L'analisi combinata di Shannon Entropy e Bit Position Analysis conferma che MD5, SHA-256 e Poseidon garantiscono un'eccellente distribuzione casuale dei bit, senza pattern sfruttabili o bias posizionali, attestando la qualità crittografica delle funzioni hash implementate.

4.9 Hash Pattern Analysis e Autocorrelazione

La combinazione tra **Hash Pattern Analysis** e **Autocorrelazione** permette di analizzare la struttura interna degli output di una funzione hash, investigando la presenza di pattern ripetuti e correlazioni interne che potrebbero compromettere la sicurezza crittografica. Entrambi i test si concentrano sulla distribuzione interna dei bit e sulla loro indipendenza, proprietà essenziali per un algoritmo hash sicuro e robusto.

4.9.1 Hash Pattern Analysis

L'**Hash Pattern Analysis** è un test statistico utilizzato per rilevare eventuali schemi ripetuti all'interno dello stesso hash. Una buona funzione hash deve produrre output privi di pattern prevedibili o ripetuti, garantendo la massima casualità e diffusione delle informazioni.

L'Obiettivo è quindi verificare che, all'interno di un singolo digest in caso di **Intra-hash pattern**, non esistano sottostringhe di bit, sufficientemente lunghe, che si ripetano, evitando così che l'output presenti regolarità strutturali che potrebbero essere sfruttate in attacchi differenziali o statistici.

1. **Definizione della dimensione della finestra di analisi**
   Si sceglie quindi una dimensione di finestra (window size) più piccola della lunghezza totale dell'hash, ad esempio 16, 24 o 32 bit. Dimensioni diverse permettono di rilevare pattern su scale differenti.

2. **Generazione delle sotto-stringhe**
   Si fa scorrere la finestra lungo l'hash, generando tutte le possibili sottostringhe consecutive di quella lunghezza. Ad esempio, per un hash di 256 bit e finestra di 16 bit, si ottengono 241 sottostringhe (256 - 16 + 1).

3. **Conteggio delle collisioni intra-hash**
   Si verifica se la stessa sottostringa compare più di una volta nello stesso hash. Ogni ripetizione oltre la prima viene conteggiata come collisione intra-hash.

4. **Calcolo del tasso di collisione intra-hash**
   Il tasso di collisione è calcolato come:

$$\text{Collision Rate} = \frac{\text{numero di collisioni intra-hash}}{\text{numero totale di finestre analizzate}} \tag{6}$$

La stessa metodologia è applicabile per l'**Inter-hash pattern**.

Se troviamo un tasso di collisione **vicino a zero** allora l'hash non presenta pattern ripetuti, mostrando una buona diffusione interna e massima casualità, come atteso in un algoritmo crittograficamente sicuro.

Altrimenti, se troviamo un tasso di collisione **elevato**, l'hash ha pattern ripetuti, indicando potenziali vulnerabilità. In un hash ideale, la probabilità che due finestre di 24 bit siano uguali è circa $\frac{1}{2^{24}}$, quindi la presenza di collisioni ripetute in finestre grandi può essere un segnale di debolezza progettuale.

Esempio pratico Supponiamo di avere un hash a 12 bit:

```
Hash: 101100110110
```

Con finestra di 4 bit, le sottostringhe generate sono:

```
1011, 0110, 1100, 1001, 0011, 0110, 1101, 1011
```

Si osservano due collisioni: "1011" appare due volte (1 e 8) e "0110" appare due volte (2 e 6). In un hash ben progettato, per finestre di dimensione significativa, il tasso di collisione dovrebbe essere vicino allo 0%.

4.9.2 Autocorrelazione

Il test di **Autocorrelazione** misura il grado di correlazione lineare tra la sequenza di bit dell'hash e sé stessa traslata di un certo lag (ritardo). Questo test valuta l'indipendenza statistica dei bit, un requisito essenziale per garantire che l'hash non presenti strutture ripetitive o prevedibili.

Si va quindi a verificare che i bit all'interno dell'hash non siano correlati tra loro, ossia che la conoscenza di un bit non fornisca informazioni sui bit vicini o su quelli a distanza fissa.

1. **Calcolo dell'autocorrelazione per ogni lag k**
   Data una sequenza binaria di lunghezza n, l'autocorrelazione a lag k si calcola come:

$$R(k) = \frac{1}{n-k} \sum_{i=1}^{n-k} b_i \cdot b_{i+k} \tag{7}$$

   dove b_i è il bit in posizione i.

2. **Ripetizione su più hash**
   Per ottenere risultati statisticamente significativi, il test viene eseguito su migliaia di hash generati da input casuali, calcolando la media dei valori di autocorrelazione per ogni lag.

- **Lag = 0**
  Deve sempre restituire autocorrelazione massima 0.5 (se normalizzato = 1), poiché ogni bit è perfettamente correlato con sé stesso.

- **Lag > 0**
  - Valori vicini a 0.25 indicano assenza di correlazione tra bit (per bit binari indipendenti, la probabilità che entrambi valgano 1 è $0.5 \times 0.5 = 0.25$)
  - Valori significativamente inferiori a 0.25 possono indicare correlazioni strutturali o pattern ripetuti, riducendo la sicurezza dell'algoritmo.

Esempio pratico Consideriamo un hash a 8 bit:

```
Hash: [1, 0, 1, 1, 0, 0, 1, 0]
```

Calcolando l'autocorrelazione per lag 1:

$$R(1) = \frac{1}{7}(1 \cdot 0 + 0 \cdot 1 + 1 \cdot 1 + 1 \cdot 0 + 0 \cdot 0 + 0 \cdot 1 + 1 \cdot 0) \tag{8}$$

$$= \frac{1}{7}(0 + 0 + 1 + 0 + 0 + 0 + 0) = \frac{1}{7} \approx 0.14 \tag{9}$$

Un valore basso, come in questo esempio, indica indipendenza tra bit consecutivi.

L'analisi combinata di **Hash Pattern Analysis** e **Autocorrelazione** consente di:

- Verificare la presenza di pattern ripetuti all'interno di un singolo hash, che ridurrebbero la diffusione e l'imprevedibilità dell'algoritmo
- Misurare l'indipendenza statistica tra i bit a diversi lag, garantendo che l'output hash non presenti strutture prevedibili sfruttabili in attacchi crittografici, come quelli usati nella crittoanalisi lineare.

Una funzione hash ideale deve mostrare un tasso di collisione intra-hash prossimo allo zero e valori di autocorrelazione coerenti con una sequenza generata pseudocasualmente e quindi significativi, garantendo priva di correlazioni interne significative.

4.10 Risultati Pattern e Correlazione

In questa sezione vengono riportati i risultati dell'analisi combinata di **Hash Pattern Analysis** e **Autocorrelazione**, effettuate per valutare la presenza di pattern ripetuti all'interno degli output hash e l'indipendenza statistica tra i bit, due requisiti fondamentali per la robustezza crittografica delle funzioni hash.

La tabella seguente riporta i risultati del test di collisione inter-hash, eseguito su finestre di diverse dimensioni (32, 24 e 16 bit) per ciascun algoritmo:

Algoritmo	Window	Collisioni totali	Windows analizzate	Collision rate
MD5	32 bit	2	97,000,000	0.000000
	24 bit	318	105,000,000	0.000003
	16 bit	93,839	113,000,000	0.000830
SHA-256	32 bit	7	225,000,000	0.000000
	24 bit	1,589	233,000,000	0.000007
	16 bit	436,894	241,000,000	0.001813
Poseidon	32 bit	7	225,000,000	0.000000
	24 bit	1,714	233,000,000	0.000007
	16 bit	434,862	241,000,000	0.001804

Come evidenziato dai risultati, i collision rate per finestre di **32 bit sono praticamente nulli** per tutti gli algoritmi, mentre per dimensioni inferiori (24 e 16 bit) si osserva un progressivo aumento delle collisioni, coerente con la riduzione dello spazio di rappresentazione. In particolare, SHA-256 e Poseidon mostrano valori molto simili tra loro, confermando un comportamento indistinguibile da quello di una funzione hash ideale.

Mentre l'analisi di autocorrelazione ha prodotto i seguenti risultati:

- **MD5**
  - Media: 0.2532
  - Massimo: 0.5000 (posizione 0)
  - Minimo: 0.2499
  - Primi 10 valori: $[0.5000, 0.2500, 0.2500, 0.2501, 0.2500, 0.2501, 0.2500, 0.2500, 0.2500, 0.2501]$
- **SHA-256**
  - Media: 0.2531
  - Massimo: 0.4999 (posizione 0)
  - Minimo: 0.2499
  - Primi 10 valori: $[0.4999, 0.2499, 0.2499, 0.2499, 0.2499, 0.2499, 0.2499, 0.2499, 0.2499, 0.2499]$
- **Poseidon**
  - Media: 0.2531
  - Massimo: 0.5000 (posizione 0)
  - Minimo: 0.2500
  - Primi 10 valori: $[0.5000, 0.2500, 0.2500, 0.2500, 0.2500, 0.2500, 0.2500, 0.2500, 0.2500, 0.2500]$

Per tutti gli algoritmi, la media dell'autocorrelazione si attesta intorno al **valore teorico atteso di 0.25**, mentre il massimo è sempre pari a $\sim$0.5 per lag = 0, come previsto dalla teoria. Non si osservano valori significativamente superiori o inferiori che possano suggerire pattern ripetitivi o correlazioni interne indesiderate.

In conclusione quindi, l'analisi congiunta di pattern e autocorrelazione conferma che:

MD5, SHA-256 e Poseidon non presentano pattern ripetuti significativi, anche per finestre di dimensioni ridotte e che tutti e tre gli algoritmi mostrano autocorrelazioni coerenti con l'indipendenza statistica dei bit, requisito fondamentale per prevenire attacchi crittoanalitici basati su correlazioni strutturali interne.

5 Conclusioni

L'analisi statistica condotta nel presente elaborato ha confermato la solidità crittografica dell'algoritmo *Poseidon*. In particolare, i test quantitativi effettuati su un ampio campione di input casuali hanno permesso di confrontare *Poseidon* con due algoritmi di riferimento – *SHA-256* e *MD5* – mettendone in luce le caratteristiche di robustezza e affidabilità.

I risultati ottenuti nei test sull'**Avalanche Effect** hanno dimostrato che *Poseidon* presenta una sensibilità ottimale alle variazioni minime dell'input, generando in media una modifica del 50.01% dei bit nell'output. Questo comportamento è perfettamente in linea con le aspettative teoriche di una funzione hash sicura, e risulta comparabile – se non lievemente più stabile – rispetto a *SHA-256*, con una deviazione standard tra le più basse osservate.

Analogamente, i test di **collision resistance**, effettuati sia sull'intero digest che su sottostringhe n-bit, hanno confermato l'assenza di collisioni anche su ampi dataset di input. L'analisi legata al **Birthday Paradox** ha mostrato che la distribuzione delle collisioni di *Poseidon* segue perfettamente l'andamento atteso teoricamente per un hash ideale, al pari di *SHA-256*.

Dal punto di vista della **uniformità dell'output**, *Poseidon* ha evidenziato una distribuzione di bit 0 e 1 pressoché identica (rapporto 0.9999), e un comportamento molto equilibrato anche nel dominio dei byte, come mostrato dai risultati del test χ^2. Il p-value ottenuto, sebbene di poco, è superiore alla soglia di significatività, confermando la validità statistica dell'ipotesi di uniformità.

L'**entropia di Shannon** calcolata per *Poseidon* (7.999993) è praticamente identica al valore massimo teorico (8.0), segno di una totale imprevedibilità dell'output. Inoltre, l'assenza di *bias* posizionali significativi nella **bit positional analysis** evidenzia come l'algoritmo non introduca schemi o regolarità che possano essere sfruttate da attacchi statistici o strutturali.

Infine, i test su **pattern hash** e **autocorrelazione** hanno confermato che *Poseidon* non presenta ripetizioni interne né dipendenze lineari tra i bit. Il tasso di collisione su finestre fino a 32 bit è stato pressoché nullo, mentre l'autocorrelazione media si è attestata sul valore atteso di 0.25, coerente con un comportamento casuale ideale.

In conclusione, l'analisi condotta dimostra come *Poseidon*, pur essendo stato progettato principalmente per l'impiego in circuiti aritmetici all'interno di sistemi ZKP, presenti un funzionamento crittografico robusto e comparabile a *SHA-256* su tutti i test analizzati. I test effettuati, infatti, non hanno evidenziato alcuna debolezza sta-

tisticamente significativa o pattern sfruttabili, confermando l'assenza di vulnerabilità evidenti.

I risultati ottenuti aprono quindi la strada a futuri approfondimenti, come ad esempio quelli legati ad analisi crittoanalitiche avanzate (differenziale, lineare).

Poseidon si conferma quindi una valida alternativa moderna alle funzioni hash tradizionali, con un design solido e prestazioni promettenti nel contesto dei protocolli avanzati di sicurezza e privacy.

Riferimenti bibliografici

[1] Saif Al-Kuwari, James H Davenport, and Russell J Bradford. Cryptographic hash functions: Recent design trends and security notions. *Cryptology ePrint Archive*, 2011.

[2] Eli Biham and Adi Shamir. Differential cryptanalysis of des-like cryptosystems. *Journal of CRYPTOLOGY*, 4(1):3–72, 1991.

[3] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for {Zero-Knowledge} proof systems. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 519–535, 2021.

[4] Lorenzo Grassi, Dmitry Khovratovich, and Markus Schofnegger. Poseidon2: A faster version of the poseidon hash function. In *International Conference on Cryptology in Africa*, pages 177–203. Springer, 2023.

[5] Howard M Heys. A tutorial on linear and differential cryptanalysis. *Cryptologia*, 26(3):189–221, 2002.

[6] Mitsuru Matsui. Linear cryptanalysis method for des cipher. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 386–397. Springer, 1993.

[7] Ilya Mironov et al. Hash functions: Theory, attacks, and applications. *Microsoft Research, Silicon Valley Campus*, 10, 2005.

[8] Bart Preneel. *Analysis and design of cryptographic hash functions*. PhD thesis, Katholieke Universiteit te Leuven Leuven, 1993.

[9] Phillip Rogaway and Thomas Shrimpton. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In *Fast Software Encryption: 11th International Workshop, FSE 2004, Delhi, India, February 5-7, 2004. Revised Papers 11*, pages 371–388. Springer, 2004.

Ringraziamenti

Eccomi arrivato alla fine di questi quasi tre anni, anni che porterò sempre con me, pieni di ricordi splendidi e momenti indimenticabili. Scrivere questi ringraziamenti mi dà la possibilità di dire a chi mi è stato accanto ciò che, forse, non dico abbastanza spesso. Non è facile racchiudere in poche righe la gratitudine che provo verso tutte le persone che mi hanno accompagnato in questo percorso. Ed è proprio in questo momento che mi rendo conto che i ringraziamenti sono la parte più difficile da scrivere.

Il primo e più grande ringraziamento va ai miei genitori, i miei primissimi sostenitori. Grazie per l'esempio che mi avete dato, per avermi insegnato cosa significa lavorare sodo, sacrificarsi per ciò che si sogna e farlo sempre con umiltà. Grazie per la pazienza, l'amore, le opportunità che mi avete dato nella vita e per avermi sempre spronato a dare il massimo in tutto ciò che facevo. Anche se non ve lo dico spesso, vi voglio bene. Spero che oggi possiate essere fieri di me e di ciò che sono diventato.

Grazie a mio fratello, Teo. Senza di te, non sarei mai venuto ogni giorno qui in università... o meglio, ci sarei venuto lo stesso, ma in autobus. Grazie per l'aiuto reciproco, tutte le litigate e le risate che abbiamo condiviso. Quante te ne ho combinate – come quella volta che ti ho buttato giù dalle scale o quando ti ho buttato la tv addosso... e chissà quante altre che è meglio non ricordare. Ma in fondo abbiamo sempre passato il tempo a scacciarci la noia l'un l'altro, a modo nostro, giocando insieme e costruendo quel legame che ci tiene uniti ancora oggi. So che ci saremo sempre l'uno per l'altro, e sono fiero di averti come fratello.

Un pensiero speciale ai miei nonni, che settimana dopo settimana, anche solo con una chiamata da lontano, non hanno mai smesso di spronarmi a costruire un futuro migliore.

Ringrazio i miei compagni, ma soprattutto amici, che ho conosciuto in questi anni. Grazie per tutto il tempo passato assieme, per ogni discussione interminabile e per tutte le prove d'esame fatte e rifatte, correggendoci a vicenda... anche se spesso finivamo per confonderci ancora di più. Porterò sempre con me il ricordo di tutte le ore trascorse nel Lab, a conoscerci meglio, raccontarci storie e condividere qualche

bevuta. Grazie per tutte le risate, ansie per gli esami, tornei di carte e pause caffè infinite, per avermi supportato e sopportato: senza di voi questo cammino non sarebbe stato lo stesso, avete reso questi anni pieni di ricordi. Ogni contributo, anche il più piccolo, è stato molto importante.

Ultimo, ma non per importanza, è giunto il momento di ringraziare me stesso, per aver creduto in me, per aver fatto tutto questo duro lavoro e per non aver mai mollato.

E ora che sono laureato in Informatica, posso finalmente usare il mio titolo per dirvi di spegnere e riaccendere quando qualcosa non funziona.