

Choosing Cryptographic Hash Functions: A Security-, Performance-, and Application-Aware Survey

Ashishkumar Gor, Ami M. Shah, Ankit P. Vaishnav
Department of Computer Engineering
Dharmsinh Desai University
Nadiad, Gujarat, India

ABSTRACT

Cryptographic hash functions are routinely selected by name—for example, “use SHA-256”—even though the suitability of a hash function depends on a richer set of requirements: collision, preimage, and second-preimage resistance; construction-level attack surface; digest and state sizes; domain separation; keyed or extendable-output functionality; short-message latency; bulk throughput; memory and energy constraints; implementation ecosystem; and standardization status. This survey develops an application-aware view of hash-function choice rather than a chronological catalogue of algorithms. It organizes modern hashing around iterative Merkle–Damgård designs, permutation/sponge designs, tree/parallel designs, and lightweight standardized designs; connects generic security bounds to multicollision, long-message second-preimage, herding, practical collision, and chosen-prefix collision results; and maps those properties to digital signatures, message authentication, software integrity, Merkle structures, content-addressed storage, embedded/IoT systems, password processing, and post-quantum hash-based signatures. The survey further introduces a Security–Performance–Application (SPA) selection model that first filters candidates using non-negotiable security constraints and then ranks feasible candidates using application-weighted performance and deployment criteria. To complement the analytical discussion, three controlled experiments examine throughput versus message size for SHA-2, SHA-3, and BLAKE2 implementations exposed through Python’s `hashlib` (with OpenSSL used where applicable), single-bit avalanche behavior, and birthday collisions on deliberately truncated SHA-256 outputs. In the measured runtime environment, SHA-256 attained the highest bulk throughput among the tested standardized fixed-output functions, while BLAKE2s led the 64-byte test; all tested functions showed mean avalanche behavior close to 50%; and the observed first-collision counts for 16–24-bit truncations closely followed the birthday expectation. A cross-size normalized analysis further demonstrates that the relative ranking changes materially with message size. These measurements are illustrative of implementation/platform effects and are not universal rankings. The survey closes with a decision procedure and open research directions for transparent, context-sensitive hash-function selection.

General Terms

Security, Algorithms, Performance, Experimentation

Keywords

Cryptographic hash function, collision resistance, preimage resistance, SHA-2, SHA-3, BLAKE2, Ascon, blockchain, IoT, benchmarking, application-aware selection

1. INTRODUCTION

A cryptographic hash function maps an arbitrary-length bit string to a fixed-length digest. In its simplest form,

$$H : \{0, 1\}^* \rightarrow \{0, 1\}^n. \quad (1)$$

The apparent simplicity of this interface hides a design space that now spans iterative compression-function constructions, permutation-based sponge functions, extendable-output functions (XOFs), tree hashing, keyed modes, and lightweight permutations. NIST currently specifies SHA-1 and the SHA-2 family in FIPS 180-4 and the SHA-3 family in FIPS 202, while SHA-3-derived functions such as cSHAKE, KMAC, TupleHash, and ParallelHash are defined in SP 800-185 [15, 16, 8]. NIST’s 2025 lightweight standard adds Ascon-Hash256 and related XOF functionality for constrained devices [21]. BLAKE2 is standardized for Internet use by RFC 7693 [19], while BLAKE3 represents a modern tree-hash design optimized for parallelism [18].

A major problem in practice is that algorithm selection is often reduced to a single question—“which hash is strongest?”—or to a single benchmark number. Neither view is sufficient. Collision resistance can dominate digital-signature safety, whereas throughput and incremental processing may dominate large-object storage. A constrained sensor may prioritize code size and energy, while a protocol deployed across heterogeneous systems may prioritize standardization and hardware acceleration. Password verification is an even stronger counterexample: a fast general-purpose hash is usually the wrong primitive; memory-hard password hashing such as Argon2 is designed for that application class [2].

This paper therefore treats hash selection as a constrained, multi-criteria decision problem. The main contributions are:

- (1) a construction-oriented taxonomy connecting Merkle–Damgård, sponge/XOF, tree/parallel, and lightweight designs;
- (2) an attack-centric synthesis connecting generic bounds to multicollision, second-preimage, herding, collision, and chosen-prefix collision results;

- (3) an application-property mapping that distinguishes what different applications actually require;
- (4) controlled experiments for throughput, avalanche behavior, and truncated-digest birthday collisions, complemented by normalized cross-size and theory-to-measurement evaluation; and
- (5) an SPA selection framework that separates mandatory security constraints from weighted implementation preferences.

2. SURVEY SCOPE AND LITERATURE-COVERAGE METHOD

This work is a *structured narrative survey*; it is not presented as a systematic literature review. The literature was organized by five evidence classes: (i) standards and government guidance, (ii) foundational construction papers, (iii) primary cryptanalysis papers, (iv) Internet specifications, and (v) contemporary algorithm specifications. Standards and algorithm status were preferentially sourced from NIST and the RFC Editor. Security-break claims were traced to primary cryptanalysis publications when practical. This evidence hierarchy is important because secondary tables often perpetuate stale claims—particularly around SHA-1 status, lightweight standardization, and the difference between a broken collision property and a broken preimage property.

The survey scope is deliberately focused on cryptographic hashing and immediately adjacent uses. Non-cryptographic hash tables and checksums are excluded except where they clarify threat-model differences. Password hashing is included only to explain why it should not be conflated with fast general-purpose hashing. Hardware synthesis results from unrelated papers are not pooled into a single numerical leaderboard because process node, FPGA family, clock target, architecture, and measurement methodology can dominate the comparison; instead, the paper gives a normalization framework and reports a controlled software experiment under a clearly stated environment.

3. SECURITY FOUNDATIONS

This section establishes the security notions used throughout the survey. It first reviews the three classical resistance properties, then explains why their effective strength depends on application context and output truncation.

3.1 Preimage, second-preimage, and collision resistance

For an ideal n -bit hash function, the classical generic work factors are commonly summarized as

$$W_{\text{pre}} \approx 2^n, \quad (2)$$

$$W_{2\text{pre}} \approx 2^n, \quad (3)$$

$$W_{\text{coll}} \approx 2^{n/2}. \quad (4)$$

The collision estimate follows from the birthday phenomenon. After hashing q independent messages into $N = 2^n$ equally likely outputs, the probability of at least one collision is approximately

$$P_{\text{coll}}(q) \approx 1 - \exp\left(-\frac{q(q-1)}{2N}\right). \quad (5)$$

Setting $P_{\text{coll}} = 1/2$ gives

$$q_{0.5} \approx \sqrt{2N \ln 2} \approx 1.1774 2^{n/2}, \quad (6)$$

while the expected number of samples until the first collision is approximately

$$\mathbb{E}[Q] \approx \sqrt{\frac{\pi N}{2}} = \sqrt{\frac{\pi}{2}} 2^{n/2}. \quad (7)$$

These generic bounds are reference points, not guarantees for a concrete construction. Structural cryptanalysis can lower the effective work factor.

3.2 Security strength is application dependent

NIST guidance distinguishes how security strength is interpreted when hash functions appear in digital signatures, HMAC, KDFs, and other constructions [5]. The most important practical lesson is that “ n -bit digest” is not synonymous with “ n bits of security” for every security goal. Collision strength for an ideal n -bit digest is approximately $n/2$ bits, while generic preimage strength is approximately n bits.

3.3 Truncation

If a hash output is truncated to t bits, the idealized generic security bounds become approximately 2^t for preimage search and $2^{t/2}$ for collisions. Consequently, truncation may be safe in one construction or protocol while being unacceptable in another. Any comparison table that lists only the native digest size but ignores deployed truncation can therefore overstate security.

4. CONSTRUCTION-ORIENTED TAXONOMY

Modern hash functions differ not only in digest length but also in how they process and combine message blocks. This section groups representative designs by construction so that later security and performance comparisons are tied to architectural differences rather than algorithm names alone.

4.1 Merkle–Damgård-style iterative hashing

The classical iterative paradigm extends a fixed-input compression function to arbitrary-length messages. Damgård’s construction principle formalized collision-preserving iteration [4]. A simplified recurrence is

$$h_i = f(h_{i-1}, M_i), \quad h_0 = IV, \quad (8)$$

with a strengthening/padding rule encoding message length. SHA-256 and SHA-512 belong to the SHA-2 family standardized in FIPS 180-4 [15]. Iterative designs have excellent deployment maturity, but naïvely using a plain iterative hash as a secret-prefix authenticator can expose length-extension behavior. HMAC avoids that misuse through a keyed construction standardized in RFC 2104 [11].

4.2 Sponge and permutation-based hashing

SHA-3 uses the KECCAK permutation in a sponge structure standardized by FIPS 202 [16]. The internal state width b is partitioned into rate r and capacity c :

$$b = r + c. \quad (9)$$

Input blocks are absorbed through the rate portion, the permutation is iterated, and output is squeezed. Capacity drives generic security bounds while rate strongly affects throughput. This structure naturally supports XOFs, and SP 800-185 extends the family with customizable and structured hashing modes [8].

4.3 Tree and parallel hashing

Tree hashing processes message chunks independently and combines intermediate chaining values. This exposes parallelism not available in a strictly serial chain. BLAKE3 explicitly adopts such a tree design and offers hashing, keyed hashing, and extendable output under a unified interface [18]. Parallelism is not automatically an application advantage: for very short messages, thread scheduling or tree-management overhead can dominate.

4.4 Lightweight standardized hashing

For constrained devices, area, code size, RAM, latency, and energy can matter as much as desktop throughput. NIST SP 800-232 standardizes Ascon-Hash256, Ascon-XOF128, and Ascon-CXOF128 alongside Ascon authenticated encryption [21]. This makes lightweight hashing a standards-backed application class rather than merely an academic collection of compact designs. Table 1 consolidates the construction families discussed above and highlights why each family is attractive in different application settings.

5. CRYPTANALYTIC LESSONS: FROM GENERIC BOUNDS TO REAL ATTACKS

Generic 2^n and $2^{n/2}$ bounds provide useful baselines, but concrete constructions can deviate from ideal random-function behavior. This section reviews representative attack classes and uses them to show how structural weaknesses translate into practical migration decisions.

5.1 Multicollisions and iterated constructions

Joux showed that iterated hash constructions permit multicollisions substantially more efficiently than a naïve random-function intuition suggests [7]. If one collision costs approximately $2^{n/2}$ work, a 2^k -way multicollision in an iterated construction can be built by chaining k collision stages at cost on the order of $k2^{n/2}$ rather than approximately $2^{n(2^k-1)/2^k}$ for a random mapping intuition. The lesson is not that SHA-2 is thereby “broken,” but that composition properties of an iterated hash require analysis beyond the three textbook notions.

5.2 Long-message second preimages

Kelsey and Schneier demonstrated second-preimage attacks on broad classes of strengthened iterated hashes for extremely long messages, exploiting expandable messages [10]. For a 2^k -block target message, their generic attack cost is approximately

$$k2^{n/2+1} + 2^{n-k+1}, \quad (10)$$

which can be far below 2^n for sufficiently long targets. This result reinforces a recurring theme: ideal bounds must be interpreted together with construction and message structure.

5.3 Herding

Kelsey and Kohno’s herding/Nostradamus attack showed another distinction between Merkle–Damgård hashes and random-oracle intuition when collision search is feasible [9]. Such results matter to applications that use a hash as a commitment to future content: collision weakness can have consequences beyond merely finding two arbitrary equal digests.

5.4 MD5 and SHA-1 as migration case studies

Wang and Yu demonstrated practical collision attacks on MD5 [22]. SHA-1 later crossed the line from theoretical weakness to a full practical collision in 2017 [20]; in 2020, Leurent and Peyrin reported a practical chosen-prefix collision and demonstrated its relevance to PGP identity certification [12]. NIST transition guidance therefore matters as operational security guidance rather than as an abstract algorithm ranking [1]. Table 2 summarizes the principal attack classes reviewed in this section and identifies the security property or application semantics each one affects.

The table emphasizes that attacks with similar computational goals can have very different application consequences; in particular, collision-oriented results are most damaging where an adversary can influence signed or committed content.

6. ANALYTICAL SECURITY AND COST MODELING OF HASH FUNCTIONS

A journal-level comparison of cryptographic hash functions should not rely only on qualitative labels such as “secure,” “fast,” or “lightweight.” These terms depend on the attack model, output size, implementation platform, and target application. This section develops a compact analytical framework for connecting generic security bounds, known cryptanalytic complexity, collision probability, implementation cost, and application-aware selection.

6.1 Generic Security Complexity

This subsection establishes the idealized work factors that serve as the baseline for later cryptanalytic and experimental comparisons. Let

$$H : \{0, 1\}^* \rightarrow \{0, 1\}^n \quad (11)$$

be an ideal cryptographic hash function producing an n -bit digest. For a generic preimage attack, an adversary is given a digest y and seeks an input x satisfying

$$H(x) = y. \quad (12)$$

For an ideal n -bit hash function, the expected generic work factor is approximately

$$C_{\text{pre}}(n) \approx 2^n. \quad (13)$$

For second-preimage resistance, the adversary is given a message x and attempts to identify a different message x' such that

$$x' \neq x, \quad H(x') = H(x). \quad (14)$$

Under the idealized random-function model,

$$C_{2\text{pre}}(n) \approx 2^n. \quad (15)$$

Collision resistance asks the adversary to find any two distinct inputs x and x' satisfying

$$x \neq x', \quad H(x) = H(x'). \quad (16)$$

Because of the birthday phenomenon, the generic collision complexity is substantially smaller:

$$C_{\text{coll}}(n) \approx 2^{n/2}. \quad (17)$$

Therefore, the ideal generic security strengths may be summarized as

$$\boxed{C_{\text{pre}} \approx 2^n, \quad C_{2\text{pre}} \approx 2^n, \quad C_{\text{coll}} \approx 2^{n/2}} \quad (18)$$

Table 1. Hash-function families.

Family	Examples	Output	Structure	Application motivation
SHA-2	SHA-256, SHA-512	Fixed	Iterative compression	Interoperability and mature software/hardware support
SHA-3	SHA3-256/512	Fixed	Sponge/permutation	Design diversity and standardized sponge hashing
SHAKE / SP	SHAKE, cSHAKE, KMAC, ParallelHash	Variable / structured	Sponge/XOF	Domain separation, variable output, MAC and parallel modes
BLAKE2	BLAKE2s/b	Fixed / keyed	ARX-based design	High software performance and keyed mode
BLAKE3	BLAKE3	Variable	Tree/parallel	SIMD/multicore throughput and verified streaming
Ascon	Ascon-Hash256/XOF	Fixed / variable	Lightweight permutation	Constrained devices and embedded systems

Table 2. Attack classes and affected properties.

Attack	Property	Goal	Application implication
Generic preimage	Preimage	Recover any message for a target digest	Digest length directly matters
Generic collision	Collision	Find any two colliding messages	Birthday bound governs ideal cost
Chosen-prefix collision	Collision	Find suffixes for attacker-selected prefixes	Particularly relevant to signatures/certificates
Multicollision	Composition	Generate many equal-digest messages	Iterated structure matters beyond pairwise collision resistance
Long-message second preimage	Second preimage	Replace a given very long target	Target length and construction can reduce cost
Herdng	Commitment-like use	Commit first, adapt message later	Collision weakness can undermine commitment semantics
Length extension	Misuse of iterative hash	Extend data while predicting hash continuation	Do not use raw $H(K M)$ as a MAC; use HMAC/KMAC

Equation (18) explains why digest length alone cannot be interpreted directly as collision-security strength. For example, an ideal 256-bit digest provides approximately 256 bits of generic preimage security but only about 128 bits of generic collision security.

6.2 Effective Security Against the Best Known Attack

The generic bounds above describe an ideal random function. Real hash constructions may exhibit cryptanalytic weaknesses that lower the effective work factor.

Let

$$\mathcal{C}_{\text{best}}(H, \alpha) \quad (19)$$

denote the estimated computational complexity of the best known attack against hash function H under attack class α , where

$$\alpha \in \{\text{pre}, 2\text{pre}, \text{coll}, \text{cp}, \text{multi}, \dots\}. \quad (20)$$

The corresponding effective security strength in bits is defined as

$$S_{\text{eff}}(H, \alpha) = \log_2(\mathcal{C}_{\text{best}}(H, \alpha)). \quad (21)$$

For an ideal n -bit hash function,

$$S_{\text{eff}}(H, \text{pre}) \approx n, \quad (22)$$

$$S_{\text{eff}}(H, 2\text{pre}) \approx n, \quad (23)$$

$$S_{\text{eff}}(H, \text{coll}) \approx \frac{n}{2}. \quad (24)$$

A useful relative security-margin measure is

$$M_{\text{sec}}(H, \alpha) = \frac{S_{\text{eff}}(H, \alpha)}{S_{\text{ideal}}(H, \alpha)}, \quad (25)$$

where $S_{\text{ideal}}(H, \alpha)$ denotes the generic ideal strength. Thus,

$$0 \leq M_{\text{sec}}(H, \alpha) \leq 1 \quad (26)$$

under the interpretation that a value close to 1 indicates that the best known attack remains close to the ideal generic bound, whereas a much smaller value indicates significant cryptanalytic degradation. This representation is useful when comparing legacy functions such as MD5 or SHA-1 with currently recommended constructions because their digest lengths alone conceal the reduction in practical collision resistance.

6.3 Exact and Approximate Collision Probability

Suppose q independently generated messages are mapped uniformly into the $N = 2^n$ possible digest values of an ideal n -bit hash function.

The probability that all q outputs are distinct is

$$P_{\text{unique}}(q, n) = \prod_{i=0}^{q-1} \left(1 - \frac{i}{2^n}\right). \quad (27)$$

Therefore, the exact probability of observing at least one collision is

$$P_{\text{coll}}(q, n) = 1 - \prod_{i=0}^{q-1} \left(1 - \frac{i}{2^n}\right). \quad (28)$$

For

$$q \ll 2^n, \quad (29)$$

the logarithmic approximation

$$\ln(1 - x) \approx -x \quad (30)$$

gives

$$P_{\text{unique}}(q, n) \approx \exp\left(-\frac{q(q-1)}{2^{n+1}}\right). \quad (31)$$

Hence,

$$P_{\text{coll}}(q, n) \approx 1 - \exp\left(-\frac{q(q-1)}{2^{n+1}}\right) \quad (32)$$

This approximation provides a direct bridge between the theoretical birthday bound and experimental truncated-digest collision studies.

6.4 The 50% Birthday-Collision Threshold

To estimate the number of sampled messages needed to reach a collision probability of approximately 0.5, set

$$P_{\text{coll}}(q, n) = \frac{1}{2}. \quad (33)$$

Using Equation (32),

$$\frac{1}{2} = 1 - \exp\left(-\frac{q(q-1)}{2^{n+1}}\right). \quad (34)$$

Therefore,

$$\exp\left(-\frac{q(q-1)}{2^{n+1}}\right) = \frac{1}{2}. \quad (35)$$

Taking the natural logarithm,

$$\frac{q(q-1)}{2^{n+1}} = \ln 2. \quad (36)$$

For sufficiently large q ,

$$q(q-1) \approx q^2, \quad (37)$$

which gives

$$q_{0.5} \approx \sqrt{2^{n+1} \ln 2}. \quad (38)$$

Equivalently,

$$q_{0.5} \approx 1.1774 \times 2^{n/2} \quad (39)$$

Table 3 illustrates this effect for selected digest sizes. The small truncated sizes in Table 3 are useful experimentally because their collision thresholds can be observed directly. Such experiments should not be interpreted as cryptanalytic breaks of the underlying full-length hash function; they illustrate the probabilistic consequences of reducing the digest space.

6.5 Avalanche and Bit-Diffusion Modeling

A desirable qualitative property of a cryptographic hash construction is that a small change in the input produces an apparently unrelated digest.

Let

$$h = H(m), \quad h' = H(m'), \quad (40)$$

where m' differs from m in exactly one input bit. Define the Hamming distance between the two n -bit digests as

$$D_H(h, h') = \sum_{i=1}^n \mathbb{I}[h_i \neq h'_i], \quad (41)$$

where $\mathbb{I}[\cdot]$ is the indicator function.

The normalized avalanche ratio is

$$A(H; m, m') = \frac{D_H(H(m), H(m'))}{n}. \quad (42)$$

For an ideal random-output model, each digest bit changes independently with probability approximately 1/2. Hence,

$$\mathbb{E}[D_H] = \frac{n}{2}, \quad (43)$$

and therefore

$$\mathbb{E}[A] = \frac{1}{2}. \quad (44)$$

If the bit differences are approximated by a binomial random variable,

$$D_H \sim \text{Binomial}\left(n, \frac{1}{2}\right), \quad (45)$$

then

$$\text{Var}(D_H) = \frac{n}{4}, \quad (46)$$

and the standard deviation of the normalized avalanche ratio is

$$\sigma_A = \frac{1}{2\sqrt{n}}. \quad (47)$$

This provides a convenient analytical benchmark for interpreting empirical avalanche measurements reported later in the experimental section.

6.6 Application Requirement Vector

Security alone does not determine the most appropriate hash function for an application. Let an application A be described by the requirement vector

$$\mathbf{r}_A = (r_c, r_p, r_s, r_T, r_L, r_M, r_E, r_\Pi, r_I, r_D), \quad (48)$$

where

- r_c denotes collision-resistance requirement,
- r_p denotes preimage-resistance requirement,
- r_s denotes second-preimage-resistance requirement,
- r_T denotes throughput importance,
- r_L denotes latency sensitivity,
- r_M denotes memory constraint,
- r_E denotes energy constraint,
- r_Π denotes parallelism requirement,
- r_I denotes interoperability/standardization requirement, and
- r_D denotes deployment or protocol-compatibility constraint.

Similarly, candidate hash function H_j may be described using a capability vector

$$\mathbf{h}_j = (s_{c,j}, s_{p,j}, s_{s,j}, T_j, L_j, M_j, E_j, \Pi_j, I_j, D_j). \quad (49)$$

The first stage of selection should enforce security feasibility rather than trading security directly against speed.

For example,

$$\mathcal{H}_{\text{sec}} = \{H_j : s_{c,j} \geq \tau_c, s_{p,j} \geq \tau_p, s_{s,j} \geq \tau_s\}, \quad (50)$$

where τ_c , τ_p , and τ_s are application-specific minimum security thresholds.

Table 3. 50% birthday-collision thresholds.

Digest size n	Generic collision scale	$q_{0.5} \approx 1.1774 2^{n/2}$
16	2^8	$\approx 3.01 \times 10^2$
24	2^{12}	$\approx 4.82 \times 10^3$
32	2^{16}	$\approx 7.72 \times 10^4$
40	2^{20}	$\approx 1.23 \times 10^6$
64	2^{32}	$\approx 5.06 \times 10^9$
128	2^{64}	$\approx 2.17 \times 10^{19}$
256	2^{128}	$\approx 4.01 \times 10^{38}$

A deployment constraint can further reduce this set:

$$\mathcal{H}_{\text{feasible}} = \{H_j \in \mathcal{H}_{\text{sec}} : D_j \geq \tau_D\}. \quad (51)$$

For an established blockchain protocol, for example, the deployment constraint may effectively require exact compatibility with a protocol-defined primitive. In a newly designed IoT system, by contrast, the designer may have considerably greater freedom after the minimum security conditions are satisfied.

6.7 Normalization of Heterogeneous Metrics

Application-aware comparison requires metrics with different physical units to be normalized before aggregation. For a benefit-type metric, such as throughput, the min–max normalized value for candidate H_j can be defined as

$$\hat{x}_j^{(+)} = \frac{x_j - x_{\min}}{x_{\max} - x_{\min}}. \quad (52)$$

For a cost-type metric, such as latency, memory consumption, energy, or hardware area, a reversed normalization is appropriate:

$$\hat{x}_j^{(-)} = \frac{x_{\max} - x_j}{x_{\max} - x_{\min}}. \quad (53)$$

Both forms map candidate performance to

$$0 \leq \hat{x}_j \leq 1, \quad (54)$$

where larger values represent more desirable outcomes.

An important limitation of min–max normalization is that the resulting scores depend on the candidate set. Therefore, the survey uses such scores for application-specific comparative analysis rather than presenting them as universal intrinsic properties of the algorithms.

6.8 Weighted Application-Fitness Score

Once mandatory security and deployment constraints have been enforced, the remaining feasible algorithms can be compared using a weighted utility model.

Let

$$\hat{\mathbf{x}}_j = (\hat{T}_j, \hat{L}_j, \hat{M}_j, \hat{E}_j, \hat{\Pi}_j, \hat{I}_j) \quad (55)$$

denote normalized application-relevant metrics. Define the application-fitness score as

$$J_A(H_j) = w_T \hat{T}_j + w_L \hat{L}_j + w_M \hat{M}_j + w_E \hat{E}_j + w_{\Pi} \hat{\Pi}_j + w_I \hat{I}_j, \quad (56)$$

with

$$w_k \geq 0, \quad \sum_k w_k = 1. \quad (57)$$

The selected candidate is then

$$H_A^* = \arg \max_{H_j \in \mathcal{H}_{\text{feasible}}} J_A(H_j) \quad (58)$$

This formulation intentionally separates two decisions:

Security determines feasibility;
application metrics determine preference.

(59)

Therefore, a highly efficient implementation cannot compensate for an algorithm that fails the application's minimum security requirements.

6.9 Sensitivity to Application Weights

A weighted score is useful only if its dependence on application priorities is made explicit.

For two candidate hash functions H_a and H_b , the preference condition is

$$J_A(H_a) > J_A(H_b). \quad (60)$$

Using Equation (56),

$$\sum_k w_k (\hat{x}_{a,k} - \hat{x}_{b,k}) > 0. \quad (61)$$

The decision boundary therefore satisfies

$$\sum_k w_k (\hat{x}_{a,k} - \hat{x}_{b,k}) = 0. \quad (62)$$

Equation (62) shows that the “best” hash can change as application priorities change. A server application may assign a large weight to throughput, while a battery-powered IoT device may assign much larger weights to energy and memory efficiency.

This sensitivity motivates a Pareto-based analysis that does not require a single fixed weighting vector.

6.10 Pareto-Dominance Analysis

A weighted score can hide legitimate trade-offs when no single set of weights is universally accepted. Pareto analysis therefore provides a complementary view that identifies candidates that are not dominated across the selected objectives. Let

$$\mathbf{f}(H_j) = (f_1(H_j), f_2(H_j), \dots, f_K(H_j)) \quad (63)$$

represent normalized objectives to be maximized.

Candidate H_a is said to dominate candidate H_b , written

$$H_a \succ H_b, \quad (64)$$

if

$$f_k(H_a) \geq f_k(H_b) \quad \forall k, \quad (65)$$

and

$$f_\ell(H_a) > f_\ell(H_b) \quad \text{for at least one } \ell. \quad (66)$$

The Pareto-efficient set is

$$\mathcal{P} = \{H_i \in \mathcal{H}_{\text{feasible}} : \nexists H_j \in \mathcal{H}_{\text{feasible}} \text{ such that } H_j \succ H_i\} \quad (67)$$

An algorithm outside $\mathcal{P}$ is unattractive under the selected criteria because another feasible candidate performs at least as well in all considered objectives and strictly better in at least one.

The existence of multiple candidates in $\mathcal{P}$ formalizes one of the central conclusions of this survey:

$$\text{No universally optimal cryptographic hash function need exist.} \quad (68)$$

Instead, the final choice depends on application priorities, deployment constraints, and the set of relevant performance objectives.

6.11 Security–Performance Efficiency

For exploratory comparison, a simple security-throughput indicator may be defined as

$$\eta_{S,T}(H_j) = S_{\text{eff}}(H_j) \times T_j, \quad (69)$$

where S_{eff} is the relevant effective security strength in bits and T_j is measured throughputs.

For a constrained implementation, one may similarly define

$$\eta_{S,E}(H_j) = \frac{S_{\text{eff}}(H_j)}{E_j}, \quad (70)$$

or

$$\eta_{T,A}(H_j) = \frac{T_j}{A_j}, \quad (71)$$

where E_j is energy consumption and A_j is hardware area. These ratios are not universal cryptographic security metrics and should not be interpreted as standards-defined quantities. Their purpose is to support comparative engineering analysis when the security threshold has already been satisfied.

6.12 Unified Analytical Selection Model

The preceding components can be summarized as a three-stage decision model.

Stage 1: Security filtering. Remove any algorithm that does not satisfy the required effective strengths:

$$S_{\text{eff}}(H_j, \alpha) \geq \tau_\alpha \quad \forall \alpha \in \mathcal{A}_{\text{required}}. \quad (72)$$

Stage 2: Deployment filtering. Require protocol, standardization, implementation, or interoperability compatibility:

$$D(H_j, A) = 1. \quad (73)$$

Stage 3: Multi-objective selection. Among feasible candidates, compute either

$$H_A^* = \arg \max J_A(H_j), \quad (74)$$

or identify the Pareto set

$$\mathcal{P}_A. \quad (75)$$

The resulting workflow is

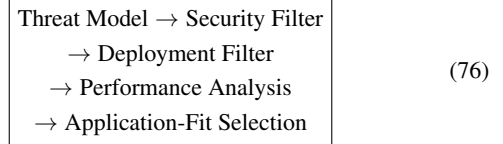


Figure 1 visualizes this three-stage process, emphasizing that performance optimization is applied only after security and deployment feasibility have been established.

The workflow prevents a high-throughput but insecure or protocol-incompatible primitive from receiving a favorable aggregate score merely because of implementation performance.

6.13 Summary of Analytical Metrics

Table 4 summarizes the principal quantities introduced in this section.

The analytical framework developed in this section provides the theoretical basis for the experimental analysis that follows. In particular, the truncated-digest experiment can be compared with Equation (39), avalanche measurements can be interpreted using Equations (44)–(47), and measured throughput can be incorporated into the application-selection model through Equations (56) and (67).

7. APPLICATION-AWARE SECURITY REQUIREMENTS

A hash function is not selected in isolation: the surrounding protocol determines which security properties and implementation constraints matter most. This section maps the preceding security analysis to representative deployment contexts ranging from signatures and software integrity to blockchain, IoT, password processing, and post-quantum signatures.

7.1 Digital signatures and certificates

Hash-and-sign systems compress potentially large messages before the public-key signature operation. Collision resistance is especially important because an attacker who can construct two semantically different documents with the same digest may attempt to transfer a signature from an innocuous document to a malicious one. The SHA-1 chosen-prefix results illustrate why collision properties must be mapped to an application’s adversarial freedom [12].

7.2 Message authentication

A hash function does not become a MAC merely by concatenating a secret key. HMAC is a dedicated construction designed around iterative hash functions [11]; KMAC provides a SHA-3-derived alternative standardized in SP 800-185 [8]. Application selection should therefore distinguish “choose a hash” from “choose a keyed construction based on a hash.”

7.3 Software and file integrity

For software distribution and archival integrity, collision and second-preimage resistance combine with ecosystem interoperability and streaming throughput. A standardized, widely implemented hash often has operational value that outweighs a modest benchmark advantage from a less ubiquitous alternative.

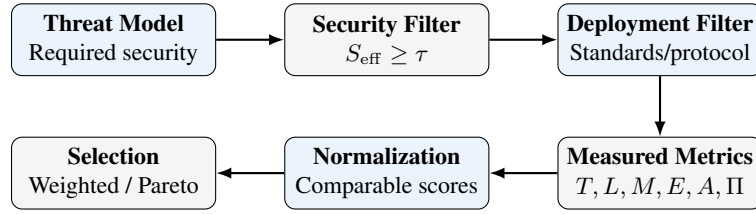


Fig. 1. Analytical workflow for application-aware hash selection.

Table 4. Analytical comparison metrics.

Quantity	Expression	Interpretation
Generic preimage complexity	$C_{\text{pre}} \approx 2^n$	Expected ideal cost of recovering a preimage
Generic second-preimage complexity	$C_{2\text{pre}} \approx 2^n$	Expected ideal cost of replacing a fixed message
Generic collision complexity	$C_{\text{coll}} \approx 2^{n/2}$	Birthday-bound collision work factor
Effective security	$S_{\text{eff}} = \log_2 C_{\text{best}}$	Security strength against the best known attack
Collision probability	$1 - \exp[-q(q-1)/2^{n+1}]$	Approximate probability of at least one collision
50% collision point	$q_{0.5} \approx 1.1774 \cdot 2^{n/2}$	Approximate birthday threshold
Avalanche ratio	$D_H(H(m), H(m'))/n$	Fraction of digest bits changed after input perturbation
Application score	$J_A(H_j) = \sum_k w_k \hat{x}_{j,k}$	Application-specific weighted suitability
Pareto set	$\mathcal{P}$	Set of non-dominated feasible hash candidates
Throughput/area	T_j/A_j	Hardware implementation efficiency
Security/energy	S_{eff}/E_j	Security-normalized constrained-device efficiency indicator

7.4 Hash Functions in Blockchain, IoT, and Resource-Constrained Systems

The requirements imposed on cryptographic hash functions vary considerably across emerging computing environments. Blockchain systems emphasize tamper-evident linking, authenticated data structures, and often very large numbers of hash evaluations, whereas Internet-of-Things (IoT) devices may operate under strict limits on energy, memory, code size, latency, and hardware area. Consequently, an algorithm that is attractive for a server-class blockchain node may not be the most appropriate choice for a battery-powered sensing device. These domains therefore provide useful examples of why hash-function selection should be application-aware rather than based on a single security or throughput ranking.

7.4.1 Blockchain and distributed-ledger applications. Cryptographic hashes perform several conceptually different functions in blockchain and distributed-ledger architectures. In the original Bitcoin design, transactions are summarized through a Merkle structure and blocks are cryptographically linked through hash-dependent block headers; the proof-of-work mechanism additionally requires repeated evaluation of a hash-based computational puzzle [14]. These uses should not, however, be treated as one generic “blockchain hashing” requirement.

A simplified block-linking relation can be represented as

$$B_i = (H(B_{i-1}), R_i, t_i, \nu_i, \dots), \quad (77)$$

where $H(B_{i-1})$ represents a digest derived from the preceding block, R_i denotes a transaction-summary root such as a Merkle root, t_i is time-related metadata, and ν_i denotes a consensus-dependent value such as a nonce. The precise header structure is protocol-specific, but the security intuition is general:

changing an authenticated historical value propagates to a different digest and therefore invalidates subsequent hash-linked commitments unless the corresponding protocol conditions are satisfied again.

For a binary Merkle tree, an internal node may be expressed as

$$v_j^{(\ell+1)} = H(v_{2j}^{(\ell)} \parallel v_{2j+1}^{(\ell)}), \quad (78)$$

and recursive application produces the root

$$R = H_{\text{tree}}(T_1, T_2, \dots, T_N), \quad (79)$$

where $T_1, \dots, T_N$ are transaction or data objects. The authenticated-tree principle is closely related to Merkle’s hash-tree constructions [13]. Given a balanced tree containing N leaves, membership verification requires only

$$O(\log_2 N) \quad (80)$$

sibling hashes rather than all N data objects. This property is especially valuable for lightweight clients, distributed storage, and scalable integrity proofs.

From a hash-selection perspective, blockchain systems may impose at least four distinct requirements:

- (1) **Collision resistance:** two distinct authenticated objects should not feasibly produce the same digest.
- (2) **Second-preimage resistance:** once an object is committed, constructing a different object that preserves the same digest should remain infeasible.
- (3) **High repeated-hash throughput:** block validation, Merkle-tree construction, indexing, and consensus-specific workloads may require a very large number of digest evaluations.

- (4) **Protocol compatibility:** unlike a newly designed standalone application, an established blockchain generally cannot replace its hash function merely because a different algorithm performs better on contemporary hardware.

The final requirement is particularly important. In cryptographic deployment, *algorithm fitness is constrained by protocol semantics*. A faster hash cannot automatically replace SHA-256, Keccak-derived hashing, or another protocol-defined primitive without affecting interoperability and potentially consensus.

Figure 2 also illustrates an important distinction: the hash function provides cryptographic binding between nodes, but *hashing alone does not provide blockchain consensus or immutability*. Consensus rules, adversarial-resource assumptions, digital signatures, network protocols, and economic or permission models contribute additional security properties. Thus statements such as “blockchain is secure because SHA-256 is secure” are technically incomplete.

7.4.2 Hash-function requirements in IoT and edge systems. IoT and embedded platforms expose a different optimization regime. A microcontroller, RFID-class device, low-power sensor, or compact hardware accelerator may have substantially tighter resource constraints than a desktop or server. Consequently, hash-function evaluation should include not only conventional throughput but also memory, implementation footprint, energy consumption, and hardware efficiency.

A simple efficiency vector for an embedded implementation H_j may be written as

$$\mathbf{e}_j = (T_j, L_j, M_j, C_j, E_j, A_j), \quad (81)$$

where T_j is throughput, L_j latency, M_j RAM usage, C_j code or ROM footprint, E_j energy per processed bit or message, and A_j hardware area where applicable.

For hardware implementations, a frequently useful efficiency quantity is

$$\eta_{\text{area}} = \frac{\text{throughput}}{\text{implementation area}}, \quad (82)$$

whereas an energy-oriented metric may be represented as

$$\eta_{\text{energy}} = \frac{\text{processed bits}}{\text{energy consumed}}. \quad (83)$$

These metrics emphasize why a desktop benchmark expressed solely in GB/s cannot determine the best hash function for an embedded node.

The relevance of lightweight hashing has increased substantially with the publication of NIST SP 800-232 in 2025. The standard specifies the Ascon-based family for constrained devices, including Ascon-Hash256, Ascon-XOF128, and Ascon-CXOF128 [21]. The standard explicitly targets environments such as IoT devices, embedded systems, and low-power sensors. Ascon therefore provides a contemporary standards-backed case study for application-aware hash selection rather than merely an academic lightweight alternative.

An IoT deployment may require hashing for several different tasks:

- firmware and secure-boot integrity;
- sensor-data authentication or commitment;
- device identification and attestation;
- authenticated data structures;
- message-authentication constructions;
- key-derivation or XOF-based functionality; and

—local evidence generation before data are transferred to an edge, cloud, or distributed-ledger service.

The primitive should therefore be selected according to the exact role. For example, a fixed-output integrity check, an XOF requirement, and a keyed authentication requirement are not interchangeable merely because all three may internally employ a permutation or hash-like construction.

7.4.3 Blockchain-IoT intersection. Blockchain-supported IoT systems combine the requirements of both domains and consequently expose a useful multi-objective hash-selection problem. Consider a sensing device that computes a digest of a measurement m_i ,

$$d_i = H(m_i \parallel \text{meta}_i), \quad (84)$$

and subsequently commits d_i to an authenticated tree or distributed ledger. The device-side operation must satisfy embedded constraints, while the ledger-facing representation must remain compatible with the distributed protocol.

The total cost of such an architecture should not be modeled as hashing cost alone. A simplified per-record cost can be written as

$$C_{\text{total}} = C_{\text{hash}} + C_{\text{comm}} + C_{\text{storage}} + C_{\text{ledger}}, \quad (85)$$

where C_{hash} is the local computational/energy cost, C_{comm} is communication cost, C_{storage} represents local or remote storage, and C_{ledger} captures protocol-dependent commitment or verification overhead.

In a deployment where communication dominates local hashing cost, one may have

$$C_{\text{comm}} \gg C_{\text{hash}}, \quad (86)$$

with the validity of this relation determined by measurement on the target device and radio stack. Under that condition, replacing one secure digest function with a slightly faster one may have negligible system-level impact if radio transmission dominates total energy. Conversely, repeated Merkle-tree construction, high-rate sensor acquisition, or local authenticated logging can make hashing cost significant. This observation reinforces the need for *end-to-end application measurements* rather than isolated primitive benchmarks. Table 5 summarizes how these priorities differ across representative blockchain, IoT, and combined deployment scenarios.

The comparison shows that protocol compatibility tends to dominate established blockchain deployments, whereas energy, memory, and code footprint become decisive in constrained devices; blockchain-backed IoT must satisfy both sets of constraints.

7.4.4 Application-aware comparison. The preceding discussion can be integrated into the Security-Performance-Application framework by augmenting the application requirement vector with deployment compatibility. Let

$$\mathbf{r}_A = (r_{\text{coll}}, r_{\text{pre}}, r_{2\text{pre}}, r_T, r_L, r_M, r_E, r_{\parallel}, r_{\text{std}}, r_{\text{dep}}), \quad (87)$$

where r_{dep} captures deployment restrictions such as protocol-defined algorithms, legacy interoperability, available hardware accelerators, and cryptographic-library support.

For established blockchain protocols, r_{dep} may behave as a hard feasibility constraint:

$$H_j \in \mathcal{H}_{\text{feasible}} \iff \text{ProtocolCompatible}(H_j) = 1. \quad (88)$$

For a newly designed IoT application, by contrast, the feasible set may be broader and optimization may place greater weight on

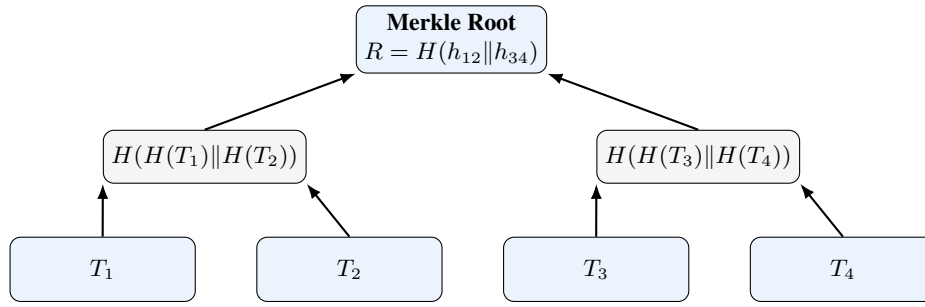


Fig. 2. Simplified Merkle-tree hash aggregation.

Table 5. Blockchain and IoT hash requirements.

Context	Security need	Main constraint	Selection implication
Block/header linking	Collision and second-preimage resistance	Protocol compatibility; repeated hashing	Protocol-defined primitive normally dominates implementation-level preference
Transaction identifiers	Collision resistance; canonical encoding	Throughput; interoperability	Digest semantics and encoding rules are as important as raw hash speed
Merkle trees	Collision resistance; unambiguous node encoding	Short-input latency; parallelism	Tree-oriented or efficiently implemented standardized hashing can be attractive
Content-addressed distributed storage	Collision resistance	Streaming throughput; digest size	Long-term interoperability and canonical representation matter
IoT firmware integrity	Second-preimage and collision resistance	Code footprint; RAM; boot latency	Standardized compact implementations are desirable
Low-power sensing	Application-dependent integrity/authentication	Energy; RAM; communication overhead	Lightweight primitives such as standardized Ascon functions become relevant
IoT XOF/domain-separated processing	Adequate cryptographic strength	Code reuse; constrained implementation	Ascon-XOF128/CXOF128 or standardized SHA-3-derived functionality may be considered according to platform requirements
Blockchain-backed IoT	End-to-end integrity across device and ledger	Energy + bandwidth + protocol compatibility	Optimize the complete path rather than the device-side hash in isolation

energy, memory, area, or code footprint:

$$H^* = \arg \min_{H_j \in \mathcal{H}_{\text{feasible}}} [w_E \hat{E}_j + w_M \hat{M}_j + w_L \hat{L}_j + w_C \hat{C}_j - w_T \hat{T}_j]. \quad (89)$$

subject to the mandatory security requirements

$$\begin{aligned} S_{\text{coll}}(H_j) &\geq \tau_{\text{coll}}, \\ S_{\text{pre}}(H_j) &\geq \tau_{\text{pre}}, \\ S_{2\text{pre}}(H_j) &\geq \tau_{2\text{pre}}. \end{aligned} \quad (90)$$

Here, the hatted quantities are normalized implementation metrics and w_k represents application-specific importance. Equation (89) is not intended to prescribe universal numerical weights; rather, it formalizes the central argument of this survey: *security constraints determine feasibility, while application and deployment constraints determine preference among secure candidates*. Figure 3 depicts

this sequence as a security filter followed by deployment and performance filters.

Figure 3 makes explicit that resource optimization is meaningful only within the subset of candidates that already satisfy the required security and deployment constraints.

The blockchain and IoT cases therefore lead to different, but complementary, selection lessons. Blockchain demonstrates that *protocol compatibility can dominate algorithm choice*, whereas constrained IoT demonstrates that *resource efficiency can dominate the ranking among otherwise secure and standardized candidates*. Their intersection shows why a single “fastest hash” or “strongest hash” conclusion is insufficient for modern heterogeneous systems.

7.5 Password processing is a different problem

Fast cryptographic hashes are intentionally efficient, which is undesirable for password verification under offline guessing. Argon2 is a memory-hard password hashing function specifically

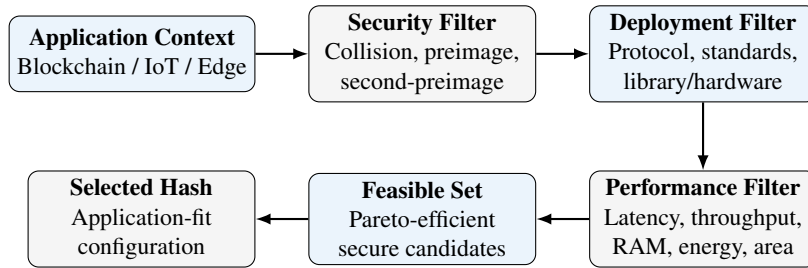


Fig. 3. Blockchain-IoT hash-selection workflow.

designed to make massive parallel guessing expensive [2]. Accordingly, a table that recommends SHA-256, SHA-3, or BLAKE2 directly for password storage is conceptually wrong even if those algorithms are cryptographically secure hashes.

7.6 Post-quantum hash-based signatures

Hash functions are central to hash-based signature constructions. NIST FIPS 205 standardizes the stateless hash-based digital signature algorithm SLH-DSA, illustrating that hash security remains important in post-quantum deployment [17]. A selection framework for such systems must additionally consider output length, number of hash invocations, implementation side channels, and quantum security margins. Table 6 summarizes how the dominant hash properties shift across the application classes discussed in this section.

The table reinforces that a primitive appropriate for one role may be unsuitable for another: for example, password verification calls for a dedicated password-hashing construction rather than a fast general-purpose digest.

8. PERFORMANCE EVALUATION METHODOLOGY

The analytical framework requires performance evidence collected under controlled conditions. This section therefore defines the benchmark scope, environment, tested algorithms, message sizes, and statistical procedures used for the empirical comparison.

8.1 Why a controlled local benchmark is preferable to a pooled leaderboard

Published software results depend on CPU model, instruction-set extensions, library version, compiler, implementation language, message size, threading, and API overhead. Hardware results additionally depend on FPGA/ASIC process and architecture. A single table copied from heterogeneous papers can therefore be numerically precise yet methodologically misleading. Accordingly, a compact controlled benchmark is used to demonstrate *how rankings change with message size and implementation context*.

8.2 Environment and algorithms

The experiment was executed under Python 3.13.5 on a Linux x86-64 containerized runtime. Python reported OpenSSL 3.5.5 for algorithms delegated to OpenSSL where applicable. The original run did not capture a stable physical CPU model, clock policy, or host-level virtualization metadata; consequently, the absolute throughput values reported below are treated strictly as environment-specific measurements rather than

hardware-characterization results. The evaluated implementations exposed through Python's `hashlib` were: SHA-256, SHA-512, SHA3-256, SHA3-512, BLAKE2s, and BLAKE2b. BLAKE3 and Ascon were excluded from the controlled speed table because comparable implementations were not available through the same standard-library interface; mixing separately optimized third-party libraries would confound the intended controlled comparison.

The tested message sizes are 64 B, 1 KiB, 64 KiB, 1 MiB, and 16 MiB. For each size, the script hashes enough repeated messages to target approximately 8 MiB of data per timing repetition (one loop for the 16 MiB case) and reports the median of three independent measurements. These settings keep the controlled study computationally modest; larger repetition counts and a named physical platform would further strengthen a future extended benchmark.

8.3 Avalanche experiment

For each function, 1000 trials generate a random 64-byte message, flip one uniformly selected input bit, and compute the percentage of output bits changed:

$$A = 100 \frac{\text{HD}(H(M), H(M \oplus e_i))}{n}, \quad (91)$$

where HD is Hamming distance. A mean near 50% is consistent with the expected diffusion of a cryptographic hash, but this empirical test is *not* a security proof.

8.4 Truncated collision experiment

To validate the birthday model empirically without making any claim against full SHA-256, SHA-256 is truncated to $t \in \{16, 20, 24\}$ bits and repeatedly hash random 128-bit inputs until the first duplicate truncated digest occurs. Thirty independent trials are run per t , and the sample mean is compared with Eq. (7).

9. EXPERIMENTAL RESULTS

This section reports the controlled measurements and interprets them in light of the analytical model. The results are presented as platform-specific observations rather than universal rankings, with separate attention to throughput, avalanche behavior, and birthday-collision scaling.

9.1 Throughput depends strongly on message size

Fig. 4 shows the measured 64-byte and 1 MiB cases. For 64-byte inputs, BLAKE2s achieved the highest measured throughput (148.8 MiB/s), followed by BLAKE2b (133.6 MiB/s) and SHA-256 (97.4 MiB/s). At 1 MiB, SHA-256 led this particular

Table 6. Application requirements and hash properties.

Application	Dominant security need	Dominant implementation need	Selection note
Digital signatures/PKI	Collision resistance; protocol-approved strength	Interoperability, standardization	Avoid algorithms with practical collision/chosen-prefix attacks
HMAC/MAC	PRF/MAC security of the full construction	Keyed API, protocol compatibility	Choose HMAC/KMAC rather than raw keyed concatenation
Software integrity	Collision and second-preimage resistance	Streaming throughput, ecosystem support	Standard support often dominates small speed differences
Content-addressed storage	Collision resistance	Bulk throughput, incremental verification	Digest length and canonical encoding matter
Blockchain/hash-linked records	Collision and second-preimage resistance	Repeated hashing, interoperability	Protocol-defined algorithms constrain replacement choices
Merkle trees/authenticated structures	Collision resistance; unambiguous encoding	Short-input latency, parallelism	Tree construction and domain separation must be considered
IoT/embedded	Adequate application-specific security strength	Energy, RAM, ROM, latency, area	Standardized lightweight functions such as Ascon are relevant
Blockchain-backed IoT	End-to-end integrity	Energy, communication, protocol compatibility	Optimize device-to-ledger system cost rather than hash throughput alone
Password storage	Offline-guessing resistance	Memory hardness, tunable cost	Use password-hashing/KDF primitives, not a plain fast hash
Hash-based signatures	Preimage/second-preimage and construction-specific properties	Many-call efficiency, side-channel control	Standardized parameter sets should drive deployment

backend at 1662.2 MiB/s, followed by BLAKE2b at 835.1 MiB/s and SHA-512 at 810.0 MiB/s. The change in ranking illustrates why “fastest” should always be qualified by message size, implementation, and platform. Figure 4 visualizes this contrast for short and 1 MiB messages, while Table 7 provides the complete median measurements across all tested sizes.

Figure 4 makes the ranking reversal easy to see, whereas Table 7 shows that the same effect persists across intermediate message sizes rather than at only two selected points.

To evaluate the complete five-size matrix without allowing the large-message absolute rates to dominate the comparison, a per-size normalized throughput is also considered:

$$\tilde{T}_{j,m} = \frac{T_{j,m}}{\max_k T_{k,m}}, \quad (92)$$

where $T_{j,m}$ is the measured throughput of algorithm j at message size m . Averaging $\tilde{T}_{j,m}$ across the five tested sizes gives 0.931 for SHA-256, 0.632 for BLAKE2b, 0.509 for SHA-512, 0.480 for BLAKE2s, 0.335 for SHA3-256, and 0.228 for SHA3-512. These values are a compact summary of this runtime only, not universal algorithm scores. Figure 5 shows the complete normalized profiles and makes the short-message/bulk crossover visible across all measured sizes.

One counterintuitive result is the strong SHA-256 bulk score relative to BLAKE2 on this environment. This should not be interpreted as a universal property of the algorithms; it is evidence that the underlying cryptographic library and CPU acceleration can reverse expectations formed from portable-C comparisons. This is precisely why application-aware selection needs implementation-aware benchmarking.

9.2 Avalanche behavior

All six tested functions produced a mean changed-bit fraction extremely close to 50% (Table 8). The standard deviation was about 3.1 percentage points for 256-bit outputs and 2.2 percentage points for 512-bit outputs, consistent with binomial concentration scaling with output length. Figure 6 provides the corresponding visual comparison.

Together, Figure 6 and Table 8 show that the observed means are close to the ideal 50% diffusion expectation. The analytical standard deviation from Eq. (47), expressed in percentage points, is 3.125 for a 256-bit output and 2.210 for a 512-bit output. The

measured standard deviations differ from these reference values by at most 0.038 and 0.034 percentage points, respectively, while the largest absolute deviation of a measured mean from 50% is only 0.084 percentage points. This agreement strengthens the diffusion check, but it is not a proof of cryptographic security.

9.3 Birthday-collision validation

The measured mean counts to the first collision were 331.3, 1415.6, and 5029.7 hashes for 16-, 20-, and 24-bit truncated SHA-256, compared with theoretical expectations 320.85, 1283.39, and 5133.57. The ratios measured/theoretical were 1.033, 1.103, and 0.980 respectively. With only 30 trials, sampling variation is expected; nevertheless, the experiment follows the $2^{t/2}$ scaling predicted by the birthday model. Figure 7 compares the measured first-collision counts directly with the theoretical expectation from Equation (7).

The relative deviations from the theoretical expectation are +3.26%, +10.30%, and -2.02% for 16, 20, and 24 bits, respectively, giving a mean absolute relative deviation of 5.19%. The close agreement in Figure 7, especially for the 16- and 24-bit truncations, supports the intended methodological point: truncation experiments reproduce birthday scaling without implying any weakness in full-length SHA-256.

10. SECURITY-PERFORMANCE-APPLICATION (SPA) SELECTION FRAMEWORK

The preceding sections show that security, implementation efficiency, and deployment constraints cannot be collapsed into a single universal ranking. The SPA framework formalizes this observation by filtering insecure or incompatible candidates first and then comparing the remaining functions using application-specific criteria.

10.1 Requirement and capability vectors

Let an application be described by a requirement vector

$$\mathbf{r}_A = (C, P, S, T, L, M, E, \Pi, I, D_{\text{dep}}, X), \quad (93)$$

where C , P , and S denote required collision, preimage, and second-preimage strengths; T throughput; L latency; M memory; E energy; Π parallelism; I interoperability/standardization;

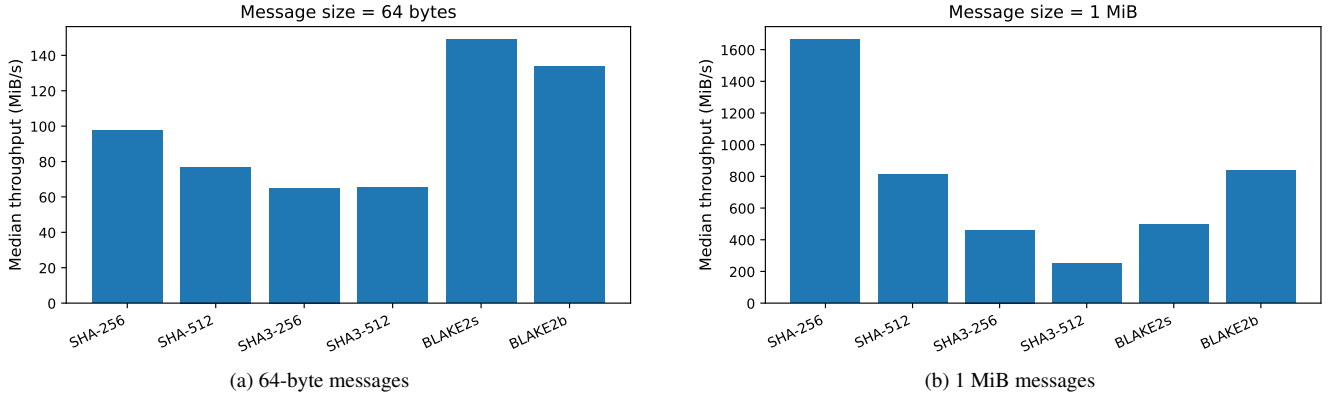


Fig. 4. Measured throughput in the test runtime environment.

Table 7. Median throughput (MiB/s).

Algorithm	64 B	1 KiB	64 KiB	1 MiB	16 MiB
SHA-256	97.4	845.9	1672.8	1662.2	1700.8
SHA-512	76.9	493.2	813.2	810.0	805.7
SHA3-256	65.0	340.5	473.3	460.8	468.6
SHA3-512	65.5	209.7	251.8	251.3	259.1
BLAKE2s	148.8	431.7	492.3	495.1	502.6
BLAKE2b	133.6	655.1	822.7	835.1	836.0

D_{dep} deployment or protocol compatibility; and X domain-separation/XOF requirements. A candidate hash H_j is represented by a capability vector c_j over compatible normalized dimensions. This notation is consistent with the analytical model of Section 6.6, where deployment compatibility is treated separately from functional requirements such as XOF support.

A simple weighted score alone is dangerous because a very fast but cryptographically unacceptable function could compensate for inadequate security. A two-stage rule is therefore used.

Stage 1: Mandatory feasibility. Define binary feasibility

$$F_j(A) = \prod_{k \in \mathcal{K}_{\text{mandatory}}} 1[c_{jk} \geq r_{Ak}], \quad (94)$$

where policy/standardization constraints can also be encoded as hard filters.

Stage 2: Preference ranking. Among feasible algorithms,

$$H^* = \arg \max_{H_j: F_j(A)=1} \sum_{k \in \mathcal{K}_{\text{soft}}} w_{Ak} u_k(c_{jk}). \quad (95)$$

with $w_{Ak} \geq 0$, $\sum_k w_{Ak} = 1$, and $u_k(\cdot)$ a normalized utility mapping. This structure prevents performance from “buying back” a failed security requirement.

10.2 Pareto analysis

When weights are disputed, report the Pareto frontier instead of one scalar winner. Candidate H_a dominates H_b if it is no worse on all selected criteria and strictly better on at least one. For constrained implementations, a typical objective set is

$$\max\{S(H), T(H), -E(H), -M(H), -A(H)\}, \quad (96)$$

where $A(H)$ denotes hardware area. A Pareto presentation is especially appropriate when comparing an interoperable SHA-2

Algorithm 1 Application-aware hash selection

Require: Application A , threat model Θ , candidate set $\mathcal{H}$

- 1: Determine mandatory security properties and minimum strengths
- 2: Remove deprecated/broken candidates for the intended use
- 3: Enforce protocol, standardization, output-length, and deployment constraints
- 4: Benchmark remaining implementations at representative message sizes on target hardware
- 5: Measure relevant resource metrics (latency, throughput, memory, energy/area where applicable)
- 6: Compute SPA utilities or determine the Pareto frontier
- 7: Perform misuse review: MAC/KDF/password/XOF/domain-separation requirements
- 8: Select H^* and record assumptions, versions, parameters, and migration plan

deployment, a sponge/XOF design, and a lightweight permutation, because their intended operating points differ.

10.3 Decision procedure

The SPA model can be operationalized as a short sequence of filters and comparisons. Algorithm 1 summarizes the procedure, beginning with mandatory security requirements and ending with a context-specific recommendation or Pareto set.

11. APPLICATION PROFILES AND SELECTION GUIDANCE

The SPA framework can be translated into practical guidance by expressing common deployment profiles in terms of dominant requirements and suitable design directions. Table 9 summarizes

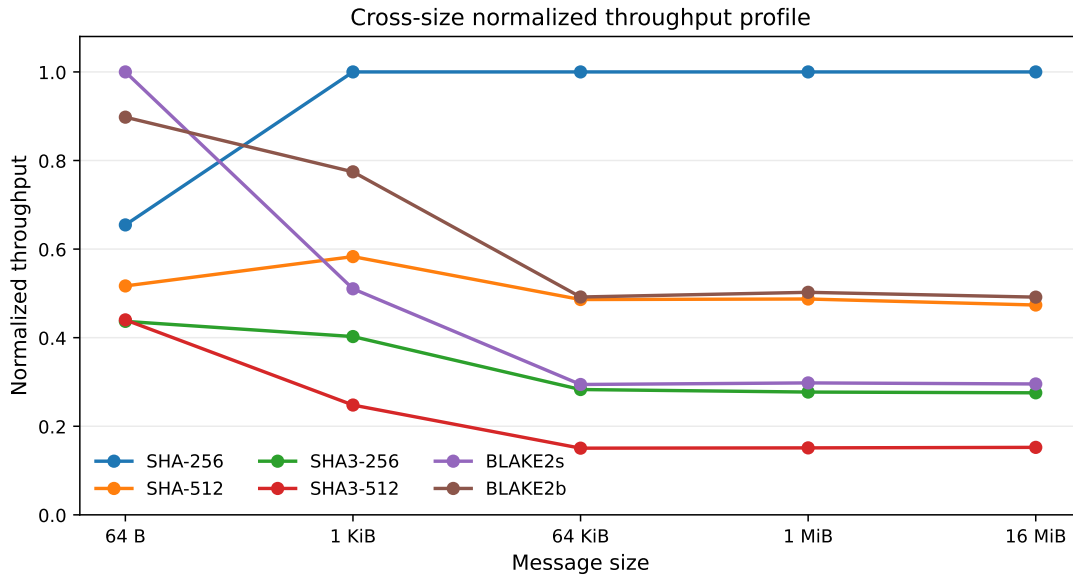


Fig. 5. Normalized throughput profiles across all tested message sizes.

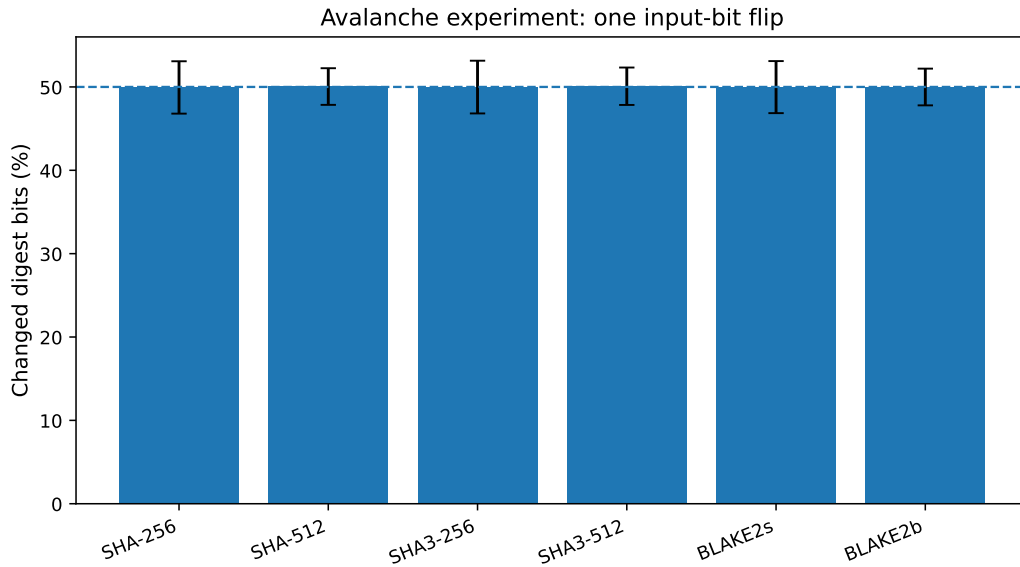


Fig. 6. Avalanche behavior under one-bit input changes.

these profiles; the entries are recommendations conditioned on the stated context rather than absolute rankings.

As Table 9 indicates, the recommendation changes with the application profile: interoperability may favor established SHA-2 deployments, XOF/domain-separation needs favor standardized sponge functions, and constrained devices may justify lightweight standardized choices.

12. COMMON COMPARISON ERRORS

Hash-function comparisons can become misleading when a single metric is treated as a complete proxy for security or efficiency. The

following recurring errors are especially important because they can reverse conclusions or encourage inappropriate deployment choices.

12.1 Treating digest length as a complete security metric

Digest length provides generic bounds, but does not summarize construction-specific attacks, truncation, protocol use, or implementation failures.

Table 8. Avalanche results.

Algorithm	Bits	Mean change (%)	SD (pp)
SHA-256	256	49.939	3.141
SHA-512	512	50.048	2.198
SHA3-256	256	49.984	3.163
SHA3-512	512	50.084	2.244
BLAKE2s	256	49.979	3.122
BLAKE2b	512	49.993	2.203

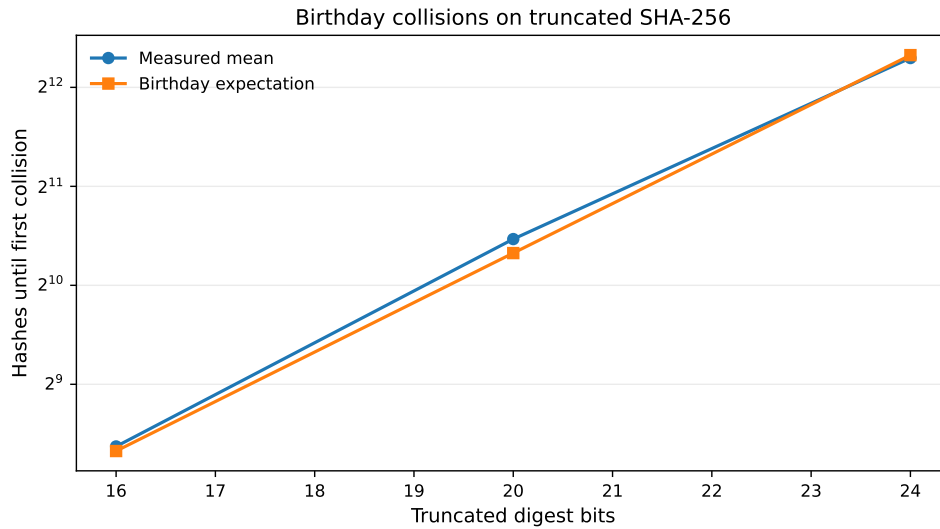


Fig. 7. Truncated SHA-256 collision counts versus birthday expectation.

12.2 Equating avalanche behavior with cryptographic security

A function can exhibit excellent 50% avalanche statistics and still be cryptographically broken. Avalanche experiments are diagnostic demonstrations of diffusion, not proofs of collision or preimage resistance.

12.3 Benchmarking one message size

Short-message performance includes call overhead and initialization; bulk hashing reflects steady-state compression/permutation throughput. Both can matter in different applications.

12.4 Mixing heterogeneous implementations

Comparing one algorithm through assembly-optimized OpenSSL, another through portable Python, and a third through a GPU kernel ranks software stacks, not algorithms. A defensible experiment either controls the implementation layer or clearly labels the comparison as end-to-end deployment performance.

12.5 Using a fast hash for passwords

Cryptographic collision/preimage security does not address offline guessing economics. Password hashing requires explicit cost amplification and preferably memory hardness [2].

13. STANDARDS AND MIGRATION LANDSCAPE

FIPS 180-4 remains the principal NIST standard for SHA-1/SHA-2 family definitions, while FIPS 202 defines SHA-3 and SHAKE [15, 16]. NIST has announced revision plans for both FIPS 180-4 and FIPS 202, which means future survey updates should distinguish the currently effective documents from planned revisions. SP 800-185 defines cSHAKE, KMAC, TupleHash, and ParallelHash [8]. SP 800-131A Rev. 2 provides transition guidance for cryptographic algorithms and key lengths [1]. The lightweight landscape materially changed with final SP 800-232 in 2025 [21]. These standards reinforce the need for surveys to timestamp algorithm-status claims.

14. OPEN RESEARCH DIRECTIONS

Several open problems remain at the intersection of cryptographic assurance, implementation efficiency, and application-specific deployment. The directions below follow directly from the gaps exposed by the analytical framework and controlled experiments.

14.1 Benchmark portability and reproducibility

Performance conclusions become more credible when the same methodology is exercised across multiple deployment classes. A useful next step is a multi-platform benchmark that runs identical test logic across x86-64, ARMv8, microcontrollers, WebAssembly, and selected hardware accelerators. Reporting should include CPU

Table 9. Hash-selection guidance.

Profile	Candidate constructions	Rationale / caveats
General interoperable integrity	SHA-256 / SHA-512	Mature standardized ecosystem; benchmark actual library/hardware
Sponge/XOF/domain separation	SHA-3, SHAKE, cSHAKE, KMAC	Standardized permutation family with flexible derived modes
High-throughput software where non-protocol hash choice is free	BLAKE2; evaluate BLAKE3 separately	Implementation and message size dominate; BLAKE3 adds tree parallelism
Blockchain/Merkle applications	Protocol-defined SHA-2, Keccak-derived, or other deployed constructions	Protocol compatibility is a hard constraint; evaluate short-input and tree-processing efficiency within the permitted design
Blockchain-backed IoT	Protocol-compatible ledger hash + suitable constrained-device primitive	Device-side energy and memory requirements must be considered jointly with ledger interoperability
Constrained/embedded standardized design	Ascon-Hash256 / Ascon XOFs	Explicitly standardized by NIST for constrained devices
Password verification	Argon2id (application-dependent parameters)	Memory-hard password hashing; do not substitute a plain fast digest
MAC	HMAC or KMAC	Use a proven keyed construction rather than ad hoc concatenation
Post-quantum hash-based signatures	Standardized hash-based signature parameter sets	Follow the signature standard; underlying hash use is construction-specific

model, clock policy, instruction sets, library version, compiler flags, thermal state, and confidence intervals.

14.2 Energy-aware and carbon-aware hashing

Throughput is only one efficiency dimension. IoT and large storage systems motivate joules/GB, energy per digest, and lifecycle cost. These measurements are currently harder to normalize than throughput but can change deployment decisions.

14.3 Application-specific domain separation

As hashes increasingly serve multiple roles—hashing, PRFs, KDFs, tree nodes, transcripts, and commitments—domain separation becomes central to misuse resistance. SHA-3-derived customizable functions provide standardized mechanisms, while other ecosystems use explicit tags or context strings.

14.4 Quantum-aware security margins

Grover's search algorithm gives a quadratic generic speedup for unstructured search, which is directly relevant to idealized preimage search, while quantum collision finding has a different query complexity [6, 3]. Application profiles should therefore state whether their security target is classical-only or includes a quantum-adversary margin rather than applying a simplistic digest-length rule.

14.5 Automatic hash selection

The SPA formulation suggests a machine-readable selection tool: inputs would include security level, protocol constraints, message-size distribution, hardware, latency/energy targets, and standardization requirements; outputs would include feasible functions, Pareto frontiers, and reproducibility metadata. The research challenge is to populate it with trustworthy, normalized, versioned measurements rather than crowd-sourced benchmark folklore.

15. THREATS TO VALIDITY

The experimental section intentionally has modest scope. It measures a single software stack in one containerized runtime environment, uses three throughput repetitions, and targets a modest data volume per repetition to keep the controlled study lightweight. The original run did not record a stable physical CPU model or host clock policy, so its absolute throughput values must not be interpreted as a reproducible hardware benchmark. It does not measure energy, hardware area, side-channel resistance, vectorized multi-buffer APIs, multicore tree hashing, or third-party BLAKE3/Ascon implementations. The results, including the normalized cross-size profile derived from the same measurements, therefore support methodological conclusions about context-dependent performance, not a universal speed ranking. The avalanche experiment measures diffusion only. The truncated-collision experiment validates birthday scaling only and must not be interpreted as evidence of a weakness in full SHA-256.

The literature coverage is representative of the paper's organizing themes but is not claimed to be exhaustive under a formal systematic-review protocol. Primary standards and landmark attacks are favored over a larger, potentially noisier bibliography.

16. CONCLUSION

Cryptographic hash functions should be selected through an application and threat-model lens, not through a one-dimensional ranking. Generic collision, preimage, and second-preimage bounds provide the starting point; construction-specific cryptanalysis determines whether those ideal assumptions remain credible; standards and protocols constrain feasible choices; and measured implementation behavior determines which feasible candidate is efficient on the target platform. The historical path from MD5 and SHA-1 collisions to SHA-2/SHA-3 deployment demonstrates the danger of treating an algorithm name as a permanent security guarantee. The emergence of standardized XOFs, lightweight Ascon hashing, tree-oriented BLAKE3, and post-quantum hash-based

signatures further broadens the decision space. The proposed SPA framework therefore applies hard security and policy filters before any performance weighting and recommends Pareto reporting when objectives conflict. The controlled experiments reinforce three practical lessons: benchmark representative message sizes, do not mistake avalanche behavior for cryptanalytic evidence, and interpret collision probability through the birthday model. Future work should expand this framework with multi-platform, energy-aware, and versioned benchmark datasets suitable for auditable automated selection.

Declarations

Funding: This research received no external funding.

Conflict of interest: The authors declare no conflict of interest.

Ethics approval: Not applicable.

Data and code availability: The experimental data and code supporting this study are available from the authors upon reasonable request.

17. REFERENCES

- [1] Elaine Barker and Allen Roginsky. Transitioning the use of cryptographic algorithms and key lengths. NIST Special Publication 800-131A Rev. 2, National Institute of Standards and Technology, 2019.
- [2] Alex Biryukov, Daniel Dinu, Dmitry Khovratovich, and Simon Josefsson. Argon2 memory-hard function for password hashing and proof-of-work applications. RFC 9106, 2021.
- [3] Gilles Brassard, Peter Høyer, and Alain Tapp. Quantum cryptanalysis of hash and claw-free functions. *ACM SIGACT News*, 28(2):14–19, 1997.
- [4] Ivan Bjerre Damgård. A design principle for hash functions. In *Advances in Cryptology—CRYPTO '89 Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 416–427. Springer, 1990. Proceedings of CRYPTO '89; conference held in 1989.
- [5] Quynh H. Dang. Recommendation for applications using approved hash algorithms. NIST Special Publication 800-107 Rev. 1, National Institute of Standards and Technology, 2012. NIST announced a planned withdrawal after migration of remaining requirements to other guidance.
- [6] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, pages 212–219. ACM, 1996.
- [7] Antoine Joux. Multicollisions in iterated hash functions. application to cascaded constructions. In *Advances in Cryptology—CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 306–316. Springer, 2004.
- [8] John Kelsey, Shu-jen Chang, and Ray Perlner. SHA-3 derived functions: cshake, kmac, tuplehash, and parallelhash. NIST Special Publication 800-185, National Institute of Standards and Technology, 2016.
- [9] John Kelsey and Tadayoshi Kohno. Herding hash functions and the nostradamus attack. In *Advances in Cryptology—EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 183–200. Springer, 2006.
- [10] John Kelsey and Bruce Schneier. Second preimages on n -bit hash functions for much less than 2^n work. In *Advances in Cryptology—EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 474–490. Springer, 2005.
- [11] Hugo Krawczyk, Mihir Bellare, and Ran Canetti. HMAC: Keyed-hashing for message authentication. RFC 2104, 1997.
- [12] Gaëtan Leurent and Thomas Peyrin. SHA-1 is a shambles: First chosen-prefix collision on SHA-1 and application to the PGP web of trust. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 1839–1856. USENIX Association, 2020.
- [13] Ralph C. Merkle. A certified digital signature. In *Advances in Cryptology—CRYPTO '89 Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 218–238. Springer, 1990. Proceedings of CRYPTO '89; conference held in 1989.
- [14] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. White paper, October 2008. Accessed 2 September 2026.
- [15] National Institute of Standards and Technology. Secure hash standard (shs). Federal Information Processing Standards Publication FIPS 180-4, National Institute of Standards and Technology, 2015.
- [16] National Institute of Standards and Technology. SHA-3 standard: Permutation-based hash and extendable-output functions. Federal Information Processing Standards Publication FIPS 202, National Institute of Standards and Technology, 2015.
- [17] National Institute of Standards and Technology. Stateless hash-based digital signature standard. Federal Information Processing Standards Publication FIPS 205, National Institute of Standards and Technology, 2024.
- [18] Jack O'Connor, Jean-Philippe Aumasson, Samuel Neves, and Zooko Wilcox-O'Hearn. BLAKE3: One function, fast everywhere. Algorithm specification, 2020. Version 20200109113900; project specification at blake3.io.
- [19] Markku-Juhani Saarinen and Jean-Philippe Aumasson. The BLAKE2 cryptographic hash and message authentication code (MAC). RFC 7693, 2015.
- [20] Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini, and Yarik Markov. The first collision for full SHA-1. In *Advances in Cryptology—CRYPTO 2017, Part 1*, volume 10401 of *Lecture Notes in Computer Science*, pages 570–596. Springer, 2017.
- [21] Meltem Sönmez Turan, Kerry McKay, Jinkeon Kang, John Kelsey, and Donghoon Chang. Ascon-based lightweight cryptography standards for constrained devices: Authenticated encryption, hash, and extendable output functions. NIST Special Publication 800-232, National Institute of Standards and Technology, August 2025.
- [22] Xiaoyun Wang and Hongbo Yu. How to break MD5 and other hash functions. In *Advances in Cryptology—EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 19–35. Springer, 2005.