Empirical Detection of Statistical Weaknesses in Cryptographic Hash Functions

A Multi-Method Black-Box Toolkit with White-Box Diffusion Validation

John & Claude | R57 Labs | April 2026

Abstract

We present a multi-method toolkit for detecting statistical weaknesses in cryptographic hash functions and block ciphers. The toolkit is primarily black-box, requiring only input-output access, and combines six independent detection methods: an aggregate statistical suite with a machine-learned meta-classifier, differential profile analysis, near-collision frequency testing, output sequence correlation, and linear approximation analysis. A supplementary white-box component measures internal state diffusion when source code is available. We validate the toolkit against synthetic weaknesses (achieving 93% detection accuracy with zero false positives in double-blind calibration) and evaluate 24 hash functions and ciphers including SHA-256, AES-128, SM3, BLAKE2b, and several non-cryptographic hashes. All cryptographic algorithms produce output indistinguishable from random oracles across all tests at 100,000 samples. Among non-cryptographic hashes, we identify a previously uncharacterized differential weakness in Jenkins one-at-a-time: while it passes all aggregate statistical tests, it exhibits significant non-uniformity in its single-bit differential profile and autocorrelation in sequential outputs. While the toolkit cannot detect kleptographic backdoors (where output is statistically perfect but contains a hidden trapdoor), it is effective at detecting implementation flaws, statistical biases, and weak differential structure. White-box diffusion analysis of six hash algorithms shows that internal mixing rates correlate with known collision vulnerability, with SHA-3 achieving full diffusion in 3 rounds while SHA-1 requires 20 rounds to reach 90%. The toolkit is released as open-source software for use in cryptographic implementation auditing and supply-chain verification.

Keywords: cryptographic analysis, hash function testing, statistical backdoor detection, differential cryptanalysis, diffusion analysis

1. Introduction

How do you know a cryptographic primitive does what it claims? If a hardware security module, a firmware update, or a software library implements SHA-256, how can you verify that the implementation faithfully follows the specification rather than introducing subtle statistical biases?

This question has practical urgency. In 2013, documents revealed that the NSA had inserted a kleptographic backdoor into the Dual EC DRBG random number generator, which was adopted as a NIST standard. Earlier, the CIA and BND operated compromised Crypto AG cipher machines for decades. DES was designed with a key size that the NSA could brute-force. History demonstrates that cryptographic standards can be deliberately weakened, and that such weaknesses may go undetected for years.

We approach this problem empirically. Rather than proving security through mathematical reduction, we build a toolkit that treats hash functions and ciphers as black boxes (or, when source code is available, white boxes) and applies multiple independent statistical tests to detect anomalies. The core insight is that no single test catches all weakness types, but a diverse ensemble of tests significantly reduces the chance that a subtle weakness escapes detection. An important boundary: this approach cannot detect kleptographic backdoors (like Dual EC DRBG) where the output is statistically perfect but predictable to a trapdoor holder. It is, however, effective at detecting implementation flaws,

deliberately weakened diffusion, statistical biases, and non-uniform differential structure.

We validate this approach using a double-blind calibration protocol: synthetic hash functions with planted weaknesses of varying types and severities, sealed behind random codenames. The detection suite runs blind, and results are compared against the sealed manifest. This prevents overfitting to known weakness patterns.

2. Methodology

The toolkit implements seven detection methods organized in three tiers: aggregate statistical tests that analyze overall output properties, differential tests that measure how outputs change when inputs change in specific ways, and structural tests that examine internal algorithm behavior. Each method is designed to detect a different category of weakness, and together they cover a broad spectrum of potential anomalies.

2.1 Aggregate Statistical Analysis

The statistical suite applies six tests to each dataset of input-output pairs: (1) **Bit correlation** measures chi-squared independence between each input bit and each output bit. (2) **Output entropy** checks per-bit and per-byte entropy for uniformity. (3) **Avalanche analysis** verifies that single-bit input changes flip approximately 50% of output bits. (4) **Conditional frequency** tests whether output byte distributions depend on input features. (5) **Mutual information** uses permutation testing (200 shuffles, 99th percentile threshold) to detect nonlinear input-output dependencies. (6) **Multi-byte interaction** tests XOR and parity combinations of input byte pairs and triples against output bits, with approximately 8,900 individual chi-squared tests per dataset.

2.2 Meta-Learned Classifier

Raw test statistics are difficult to interpret because thresholds depend on sample size, output width, and the number of tests performed. We train a logistic regression classifier on 23 features extracted from the statistical suite output, including sample-size-normalized chi-squared values (chi2/N), width-normalized significant-pair fractions, phi coefficients, and signal strength scores. We chose logistic regression over random forest or gradient boosting because the decision boundary is fully interpretable: each feature's learned weight indicates its contribution to the weakness score. The classifier is trained on 288 synthetic variants across four output widths (64, 128, 160, 256 bits) and three sample sizes (2K, 10K, 50K), achieving 90.6% training accuracy with F1 = 0.914. A learned decision threshold (0.40) replaces hand-tuned composite scoring.

The three most predictive features by learned weight are: conditional frequency signal strength (+1.16), normalized frequency chi-squared (+1.00), and bit correlation signal strength (+0.81). Multi-byte interaction (+0.77) and bit correlation phi coefficient (+0.60) round out the top five. Notably, avalanche deviation and mutual information have near-zero weights, suggesting they contribute little beyond what the other features capture.

2.3 Differential Profile Analysis

Where aggregate tests ask "does the output look random?", differential tests ask "does the output *change* randomly?" We implement four differential tests: (1) **Single-bit differential matrix** measures the probability that each output bit flips when each input bit is flipped, testing for deviation from the ideal 0.5. (2) **Byte-level differential** measures the entropy of output byte XOR deltas when input bytes are changed. (3) **Hamming distance profile** tests whether output Hamming distances are correctly distributed for input pairs at fixed distances (1, 2, 4, 8, 16 bits). (4) **Differential bit independence** tests whether output bit flips are correlated with each other, conditioned on a single input bit flip.

2.4 Extended Analysis

Three additional tests target specific structural properties: (1) **Near-collision frequency** measures whether input pairs differing by 1 bit produce outputs that share more prefix bits than expected (2^{-k} for k-bit prefix), comparing against random-pair controls. (2) **Sequence correlation** hashes sequential counter-mode inputs and tests for autocorrelation, non-ideal Hamming distances, and run-length anomalies in consecutive outputs. (3) **Cycle and fixed-point detection** iterates the hash function (using Floyd's algorithm) to detect short cycles or convergence patterns.

2.5 Linear Approximation Analysis

Inspired by Matsui's linear cryptanalysis, we test whether any linear combination of input bits correlates with any linear combination of output bits. We test six levels of mask complexity: (1) all 1x1 bit pairs exhaustively (typically 1,280-16,384 tests depending on output width), (2) 200 sampled 2-bit input masks x each output bit (~6,400 tests), (3) each input bit x 200 sampled 2-bit output masks (~6,400 tests), (4) 5,000 sampled 2x2 combinations, (5) 3,000 sampled 3-bit input masks x single output bits, and (6) byte-parity structured masks covering all input byte pairs. For a perfect hash, every linear approximation should have bias indistinguishable from 1/sqrt(N). A signal is flagged when the maximum observed bias exceeds 2.5x the noise floor (4/sqrt(N)).

2.6 Internal State Diffusion (White-Box)

For algorithms where source code is available, we instrument pure Python implementations to record the internal state after each compression round. By measuring the Hamming distance between internal states of original and 1-bit-flipped inputs at each round, we construct a *diffusion curve* showing how quickly a single-bit change propagates through the internal state. A well-designed hash reaches approximately 50% state-bit diffusion within a few rounds; slow diffusion creates the differential paths that enable collision attacks.

3. Calibration and Validation

3.1 Double-Blind Synthetic Calibration

We validate the detection pipeline using a double-blind protocol. A generator creates hash function variants: some clean (seeded SHA-256), some with planted weaknesses at random severities. Four weakness types are supported: bit correlation (specific input-output bit dependencies), frequency bias (input-conditional output byte distributions), weak avalanche (reduced diffusion in leading bytes), and subset leak (output preserving functions of input byte subsets). Each variant receives a random codename; the weakness mapping is sealed with a SHA-256 verification hash. Detection runs blind.

Results on 10,000-sample calibration batches: the statistical suite correctly classified 22/22 variants (zero false positives, zero false negatives). The meta-learner achieved 93.3% accuracy on held-out data with clean variants scoring 0.10-0.24 probability and weakened variants scoring 0.92-1.00, demonstrating strong separation. The targeted neural probe independently confirmed 9/10 weakened variants with zero false positives.

3.2 CRC32 Baseline

CRC32 provides an important baseline: its weakness is algebraic (XOR-linearity), not statistical. As expected, the aggregate statistical suite and meta-learner classify CRC32 as clean (probability 0.11), confirming they do not false-positive on algebraic weaknesses. However, the differential profile analysis detects CRC32 strongly (single-bit differential: 1.000, bit independence: 1.000), as does sequence correlation (0.945 autocorrelation). This validates that different methods detect different weakness categories.

4. Results

We evaluated 24 algorithms across all detection methods at 100,000 samples per dataset (statistical suite and extended analysis used three input distributions: random strings, dictionary words, and sequential counters). Table 1 shows the complete detection matrix.

Algorithm	Type	Bits	Stat Suite	Diff Profile	Extended	Linear Approx	Diffusion
SHA-256	Hash	256	0.18	-	-	-	16.0%
SHA-512	Hash	512	0.26	-	-	-	12.9%
SHA-1	Hash	160	0.20	-	-	-	4.4%
SHA-224	Hash	224	0.19	-	-	-	n/a
SHA-512/256	Hash	256	0.23	-	-	-	n/a
SHA-3 (256)	Hash	256	0.20	-	-	-	100%
SHA-3 (512)	Hash	512	0.26	-	-	-	n/a
BLAKE2b	Hash	256	0.21	-	-	-	n/a
BLAKE2s	Hash	256	0.22	-	-	-	n/a
MD5	Hash	128	0.23	-	-	-	9.1%
SM3	Hash	256	0.19	-	-	-	31.7%
AES-128	Cipher	128	0.21	-	-	-	n/a
ChaCha20	Cipher	256	0.21	-	-	-	n/a
SM4	Cipher	128	0.23	-	-	-	n/a
Camellia	Cipher	128	0.25	-	-	-	n/a
Adler-32	Chksum	32	P	S	S	S	n/a
CRC32	Chksum	32	0.23	S	S	-	n/a
DJB2	Non-C	32	S	S	S	-	n/a
FNV-1a (32)	Non-C	32	S	S	S	-	n/a
FNV-1a (64)	Non-C	64	S	S	S	-	n/a
Pearson-64	Non-C	64	S	S	S	S	n/a
Jenkins OAT	Non-C	32	0.15	S	S	-	n/a
MurmurHash3	Non-C	32	0.23	-	-	-	n/a
SipHash-2-4	PRF	64	0.23	-	-	-	n/a
Oracle-256	*Control*	256	0.21	-	-	-	n/a
Oracle-128	*Control*	128	0.18	-	-	-	n/a

*Table 1. Detection matrix across 24 algorithms, 7 oracle controls, and 7 methods. **Legend:** **S** = signal detected, **P** = partial (some input distributions only), 0.xx = meta-learner probability (clean; threshold 0.40), - = clean on non-statistical tests. Diffusion column shows the percentage of state bits affected by a single input bit change at round 5 of the compression function (higher = faster diffusion). Oracle controls are synthetic random functions matched to each output width.*

4.1 Cryptographic Algorithms

All 15 cryptographic algorithms (11 hashes and 4 ciphers) produced clean results across every detection method. Meta-learner probabilities ranged from 0.13 to 0.26, within the same band as their matched random oracle controls (0.12-0.22). No algorithm showed any anomaly on any input distribution. SHA-256, AES-128, SM3, BLAKE2b, ChaCha20, Camellia, and SM4 are all statistically indistinguishable from perfect random functions at our sensitivity level.

4.2 Non-Cryptographic Hashes

The non-cryptographic hashes reveal a natural detection hierarchy. At the bottom, Adler-32, DJB2, FNV-1a, and Pearson-64 are detected by most or all methods. These algorithms have fundamentally insufficient diffusion, and their weaknesses manifest across multiple statistical dimensions. CRC32 occupies a unique position: invisible to aggregate statistical analysis (its output distribution is uniform) but clearly detected by differential and sequence correlation tests due to its XOR-linearity.

4.3 Jenkins One-at-a-Time: A Case Study

Jenkins one-at-a-time provides the strongest validation of the multi-method approach. This widely-used non-cryptographic hash passed every aggregate statistical test, producing output indistinguishable from a random oracle under bit correlation, entropy, frequency, mutual information, and interaction analysis. The meta-learner classified it as clean (probability 0.14-0.15 across all distributions).

However, differential profile analysis detected significant non-uniformity in Jenkins' single-bit flip probabilities (signal strength 1.000), and sequence correlation analysis found significant autocorrelation in consecutive outputs (signal strength 1.000). Jenkins' weakness is not in what it outputs, but in how its output changes.

The structural reason is Jenkins' simple shift-and-add construction with no final mixing pass. The function processes input bytes sequentially, accumulating via addition and XOR with shifted copies of the state. Later input bytes undergo fewer rounds of mixing before the final output, creating position-dependent differential structure. The Wikipedia article on Jenkins hash includes an avalanche diagram showing weak mixing in the final byte; our differential matrix and sequence correlation analysis provide a more complete characterization of this property.

This finding has limited practical security implications — Jenkins is a hash table function, not a cryptographic primitive, and its intended use case does not require differential uniformity. However, it serves as a methodological proof of concept: structural weaknesses can exist that are invisible to aggregate statistical analysis but detectable by differential methods. This validates our multi-method approach and suggests that any serious evaluation of cryptographic implementations should include differential testing, not just output distribution analysis.

4.4 Well-Designed Non-Cryptographic Hashes

MurmurHash3 and SipHash-2-4 are completely clean across all seven detection methods, indistinguishable from random oracle controls. This is notable because neither claims cryptographic security. MurmurHash3 was specifically designed for good avalanche properties in hash table applications; SipHash-2-4 was designed as a cryptographic PRF. Their clean results confirm that achieving statistical quality is a design choice, not an inherent property of complexity.

4.5 Internal State Diffusion

A note on interpretation. Diffusion rate is a descriptive metric about internal mixing behavior, not a predictive security metric. Fast diffusion does not imply greater security, and slow diffusion does not guarantee vulnerability. Actual collision resistance depends on many factors — message schedule complexity, round function algebraic structure, state size — that a diffusion curve does not capture. We present these measurements to characterize implementation behavior, not to rank security.

With that caveat established: white-box diffusion analysis of six hash algorithms reveals clear differences in their internal mixing rates. We instrument pure Python implementations of MD5, SHA-1, SHA-256, SHA-512, SM3, and SHA-3 to record internal state after each compression round. Table 2 shows the results, sorted by early-round diffusion rate.

Algorithm	Design	State	Rounds	Diffusion at Round 5	Rounds to 90%	Known Collision Attacks
SHA-3 (256)	Sponge	1600b	24	**100%**	**3**	None known
SM3	M-D	256b	64	**31.7%**	**12**	None known
SHA-256	M-D	256b	64	17.1%	16	None known
SHA-512	M-D	512b	80	12.9%	**16**	None known
MD5	M-D	128b	64	9.1%	18	Wang et al. 2004
SHA-1	M-D	160b	80	4.4%	20	Stevens et al. 2017

Table 2. Internal state diffusion comparison across six hash algorithms. "Diffusion at Round 5" shows the percentage of ideal Hamming distance (state_bits / 2) achieved after 5 compression rounds. Higher indicates faster mixing. "Rounds to 90%" shows how many rounds are needed to reach 90% of ideal diffusion. Algorithms sorted by early-round diffusion rate (best to worst). M-D = Merkle-Damgard construction.

SHA-3's sponge construction achieves full diffusion in just 3 rounds across its 1600-bit state — by round 5 it is at 100%. With 24 total rounds, this represents a 21-round security margin. At the other extreme, SHA-1 reaches only 4.2% diffusion by round 5, requiring 20 of its 80 rounds to reach 90%. Among the Merkle-Damgard family, SM3 stands out: despite sharing SHA-256's state size and round count, it achieves nearly double the early-round diffusion (31.7% vs 16.0% at round 5). This suggests the Chinese designers made deliberate choices to accelerate mixing in SM3's round function.

As noted above, these measurements do not imply that SHA-3 is 'more secure' than SHA-256 — both are unbroken and considered secure by the cryptographic community. SHA-1's slow early diffusion contributed to its vulnerability, but the actual collision attacks exploited specific properties of SHA-1's message schedule and round function algebraic structure. MD5 was broken 13 years before SHA-1 despite comparable diffusion metrics, largely because MD5's simpler message schedule made differential path construction significantly easier. Direct comparison across architectures requires care: algorithms differ in state size, round structure, and total round count. The early-round diffusion percentage normalizes for state size and provides a more meaningful comparison than raw round counts.

5. Limitations and Threat Model

Our toolkit detects a specific category of weakness: statistical anomalies in the input-output mapping and differential structure of hash functions. Table 3 maps our detection capability against the known taxonomy of cryptographic backdoors.

Backdoor Type	Example	Detectable by Toolkit?	Requires
Statistical bias in output	Synthetic calibration variants	**Yes**	Black-box I/O pairs
Weak differential structure	CRC32, Jenkins OAT	**Yes**	Black-box I/O pairs
Slow internal diffusion	MD5, SHA-1 early rounds	**Yes**	White-box implementation
Kleptographic (hidden trapdoor)	Dual EC DRBG	**No**	Parameter analysis
Reduced keyspace	DES 56-bit, A5/1	**No**	Specification review
Algebraic/structural	GOST key schedule	**Partial**	Algebraic analysis

Table 3. Backdoor taxonomy and detection coverage. The toolkit detects statistical bias, differential structure weaknesses, and slow internal diffusion. It does not detect kleptographic backdoors (where output is statistically perfect but predictable to a trapdoor holder) or reduced-keyspace attacks.

The most sophisticated known backdoor, Dual EC DRBG, is kleptographic: its output is statistically indistinguishable from random. No amount of input-output testing can detect this class of weakness. Detection requires analysis of the mathematical relationship between algorithm parameters, which is a fundamentally different problem. Similarly, reduced-keyspace attacks (like DES's 56-bit key) affect security without any output anomaly.

Our null result on cryptographic algorithms should be interpreted carefully. We can state: at 100,000 samples across three input distributions, no tested algorithm exhibits detectable statistical bias, differential non-uniformity, sequence correlation, linear approximation bias, or abnormal internal diffusion. We cannot state that these algorithms are free of all weaknesses.

6. Related Work

The NIST Statistical Test Suite (SP 800-22) provides a standard battery for evaluating random number generators. Our aggregate statistical suite is conceptually similar but specifically targets hash function input-output relationships rather than output-only randomness. Burp Suite's Sequencer tool evaluates token randomness using FIPS tests. Our toolkit extends this by testing the input-output relationship (information leakage), not just output quality.

Matsui's linear cryptanalysis (1993) and Biham-Shamir differential cryptanalysis (1990) are foundational techniques for analyzing block ciphers. Our linear approximation and differential profile tests are black-box empirical versions of these analytical methods, applicable without knowledge of the algorithm's internal structure.

The Strict Avalanche Criterion (SAC) and Bit Independence Criterion (BIC), introduced by Webster and Tavares (1985), formalize the diffusion properties we test. Our single-bit differential matrix and differential bit independence tests are direct empirical evaluations of SAC and BIC respectively.

7. Conclusion and Future Work

We demonstrate that a multi-method detection toolkit, validated through double-blind calibration, can reliably detect statistical weaknesses in hash functions while maintaining zero false positives on cryptographic primitives. The toolkit's layered approach provides defense in depth: weaknesses invisible to aggregate statistical analysis (Jenkins OAT, CRC32) are caught by differential and sequence correlation tests. No single method suffices.

Practical applications include supply-chain verification of cryptographic implementations, regression testing for hardware security modules, and automated screening of proprietary encryption libraries. The toolkit requires only input-output access for black-box analysis, with optional white-box instrumentation when source code is available. An accompanying command-line tool (*cryptid*) packages the full pipeline for practical use.

For integration into development workflows: the *quick* test level (statistical suite and meta-learner) runs in under 5 seconds at 5,000 samples, making it suitable for CI/CD pipelines on every commit. The *standard* level (adding differential profile) completes in approximately 10 seconds. The *full* level (all tests at 100,000 samples) takes several minutes and is better suited as a nightly or release-gate check.

Future work includes: algebraic degree testing for Boolean function complexity, extension to elliptic curve parameter suspicion testing to address kleptographic backdoors, and TLS/SSH implementation fingerprinting to detect non-standard cryptographic behavior in network protocols.

The complete toolkit source code, calibration data, and raw results are available at: **github.com/r57-labs/cryptid**

Contact: mail@r57labs.com