LibFWHT: From Exact Walsh Spectra to Key Dependence in Differential-Linear Correlations

Hosein Hadipour[1], Saleh Khalaj Monfared[2], Jens Alich[1] and Jan Vorloeper[1]

[1] Ruhr University Bochum, Bochum, Germany
[2] Worcester Polytechnic Institute (WPI), Worcester, USA

Abstract. The Walsh-Hadamard transform (WHT) computes the correlation of a Boolean function with every input parity function at once. Cryptanalytic workflows transform large arrays repeatedly, in batches. However, available tools specialize in particular datatypes, platforms, or ecosystems, and none combines what a portable search needs. We present LibFWHT, an open-source C library for dense Walsh-Hadamard transforms, with Python bindings and a command-line interface. It includes vectorized, multicore, and graphics-processor backends, batch operations, and Boolean and substitution-box helpers. With one central processing unit (CPU) core, LibFWHT is 2.5 to 3.3 times faster than FFTW and up to 3.7 times faster than sboxU on batched Boolean spectra. On an NVIDIA H200 its device-resident graphics-processor backend reaches 1.8 trillion operations per second and runs the same Boolean workload about 600 times faster than the CPU-only sboxU. We demonstrate LibFWHT in linear and differential-linear cryptanalysis of SIMON-32, KATAN-32, BEANIE, KeeLoq, and RC5-16, all with 32-bit blocks. We report the first exact differential-linear distinguishers for KATAN-32, BEANIE, KeeLoq, and RC5-16, whose correlations are measured over the complete codebook rather than estimated from round-by-round trails. We then analyze how these correlations depend on the key, using the geometric approach to cryptanalysis. Writing a fixed-key correlation as a signed sum over key masks turns key dependence into a question of which masks survive. We address this question by combining a mixed-basis trail search with exact measurement on sampled keys. The search proposes responsible round-key bits, and only the measurement determines whether those bits explain the dependence. The procedure separates three outcomes: dependence concentrated on a few key bits, cancellation at the enumerated leading orders, and dependence only weakly explained by the tested short list. Distinguishers are normally reported as a single correlation, and key-recovery complexities are computed from that number. We show that relying on this summary is unsafe. For one of our distinguishers, a quarter of the keys belong to a weak class. Its root-mean-square correlation implies a data requirement about 40 times that estimated from the overall root-mean-square correlation. Key dependence therefore belongs in the analysis, and we report distributions over keys rather than single numbers.

Keywords: Walsh-Hadamard transform, fast Walsh-Hadamard transform, symmetric-key cryptanalysis, geometric approach to cryptanalysis, GPU, SIMD

1 Introduction

The Walsh-Hadamard transform (WHT) is a fundamental tool in symmetric-key cryptanalysis. For a Boolean function, it computes the correlation with every linear parity function

E-mail: hosein.hadipour@rub.de (Hosein Hadipour), skmonfared@wpi.edu (Saleh Khalaj Monfared), jens.alich@rub.de (Jens Alich), jan.vorloeper@rub.de (Jan Vorloeper)

at once. It replaces a table of binary dot products with one structured computation.

The WHT is a frequently used subroutine of many cryptanalytic tasks. Linear cryptanalysis [Mat93] uses linear approximation tables (LATs), which record correlations for component functions. Differential cryptanalysis [BS90] uses difference distribution tables (DDTs), which count how input differences map to output differences. Differential-linear cryptanalysis [LH94] combines a differential relation with a linear relation. The commonly used differential-linear connectivity table (DLCT) [BDKW19, HDE24] computes output correlations for a fixed input difference with one WHT of an output-difference histogram. Recent methods also use WHTs in surrogate-differentiation searches and round-based differential-linear correlation calculations [DDK+23, HNW26].

For a Boolean function on n input bits, computing all 2^n coefficients independently needs $\Theta(2^{2n})$ operations. The fast Walsh-Hadamard transform (FWHT) lowers this cost to $n2^n$ operations, carried out as n butterfly passes. Each pass reads and writes the full 2^n-element array. At large sizes, computing the dense FWHT is mainly limited by memory bandwidth. An effective implementation must therefore match the butterfly structure to the hardware memory hierarchy.

Related Work and Positioning.

Software for Walsh-Hadamard transforms. Existing tools address individual parts of the computational problem. `FFTW` offers an indirect WHT route through multidimensional size-two discrete Hartley transforms [FJ05, FJ20]. `FFHT` targets single-thread x86 floating-point transforms, `DaoFHT` targets NVIDIA CUDA and PyTorch half-precision (`fp16`) workflows, and `Hadamard.jl` serves the Julia ecosystem [FAL15, Dao23, Jul14]. `sboxU` provides native Boolean Walsh transforms within a broader discrete-function analysis toolkit [PAB+26], but it executes on the host only and offers no graphics processing unit (GPU) backend. However, these tools do not provide the same combination of dense integer transforms, Boolean conventions, multicore and graphics-processor execution, and an embeddable C application programming interface for cryptanalytic search tools. We defer matched performance comparisons and their conditions to Section 5.

Composed and exact correlations. Most published distinguishers are obtained by composing estimates taken from smaller parts of a cipher. A differential-linear (DL) analysis splits the cipher into an upper part that propagates a difference and a lower part that propagates a mask, then combines the two measured quantities [LH94, BDKW19, HDE24]. Automated searches extend this composition to large sets of trails [ZWH24, CXZQ25]. Composition of this kind is a heuristic, because it assumes that the parts behave independently, and the resulting value can differ from the correlation that the cipher actually has under a fixed key [Goh19]. When the block is small enough, the assumption can be dropped instead of examined: one evaluates the entire codebook under a fixed key and reads the exact correlation from it. What stands in the way is then arithmetic rather than modelling, which is why a fast transform library is the component that makes this route practical.

Key dependence of distinguishers. A second line of work asks what a correlation does as the key varies. Ankele and Kölbl studied the analogous key dependence of differential probabilities, observing a bimodal distribution for SPECK-64 and evidence of a large weak-key class for Midori-64 [AK18]. Beyne introduced a geometric approach in which a correlation under a fixed key is exact, and Beyne and Rijmen showed that a fixed-key differential probability decomposes into terms whose signs depend on the round keys [Bey21, BR22]. Hu et al. extended the construction so that different parts of a cipher can be described in different bases, which is what a DL distinguisher needs [HZC+25]. Tezcan, Leander, and Hadipour tested published DL distinguishers over random keys with GPU experiments. Several claims failed when their theoretical assumptions did not hold, whereas the experimentally accessible distinguishers of Hadipour, Derbez, and

Eichlseder remained valid in practice [TLH25]. The GPU experiments measured key-dependent behavior, whereas the geometric approach explains it. However, to the best of our knowledge the two have not been brought together on measured whole-cipher distinguishers.

The target ciphers. Cipher-specific DL studies of SIMON-32 were given by Chen and Zhang [CZ18] and by Zhang et al. [ZLZ23], and automated searches for SIMON-like ciphers were developed by Zhou et al. [ZWH24] and Chen et al. [CXZQ25]. Shi et al. reported round-reduced distinguishers for KATAN-32, including an 84-round characteristic and a trail hull on the same external masks [SHSS16]. BEANIE was proposed recently, and its designers analyzed it only inside the restricted attacker model they define [GHM+25]. KeeLoq has been studied since the slide-algebraic attack of Bogdanov, whose linear relation is stated for a 32-cycle internal-state approximation rather than for the full cipher [Bog07]. For RC5, Kaliski and Yin proposed linear approximations whose estimated biases Selçuk subsequently compared with exhaustive measurements over random keys [KY95, Sel00]. However, we are not aware of exact DL distinguishers for KATAN-32, BEANIE, KeeLoq, or RC5-16, that is, distinguishers with an explicit input difference and output mask whose correlation is measured over the full codebook.

Our Contribution.

This paper presents `LibFWHT`[1], a C99 library for Walsh-Hadamard transforms. It accelerates existing symmetric-key cryptanalysis methods rather than defining a new cryptanalytic model. Faster tools let researchers run larger experiments. For small block ciphers, `LibFWHT` treats a fixed-key cipher as an input-output function. It then computes component-function Walsh spectra directly. This approach avoids combining estimates from smaller cipher parts under independence assumptions [Goh19]. Measuring a distinguisher on many keys then raises a question that one summary value cannot answer, namely which key bits its strength depends on. We answer that question for four distinguishers with the geometric approach, and we report what the answers imply for how measured distinguishers should be presented. We apply `LibFWHT` here to linear and DL distinguishers, but the transform serves many further cryptanalytic tasks, among them key recovery and S-box analysis, which is why we treat portable multi-core and GPU execution as infrastructure rather than as an optional feature.

The contributions of this paper are as follows.

- We provide `LibFWHT`, a portable library combining vectorized, multicore, and GPU execution with batch transforms and specialized Boolean and substitution-box (S-box) routines. On a single CPU core, it is 2.5–3.3 times faster than `FFTW`'s indirect route and twice as fast as `sboxU` for a Boolean spectrum of 2^{29} entries. Its CUDA backend reaches 1.82 trillion operations per second on an NVIDIA H200 for device-resident 32-bit integer transforms.

- We use `LibFWHT` for a full-codebook study of SIMON-32, KATAN-32, BEANIE, KeeLoq, and RC5-16. For each fixed key, we compute the entire codebook and obtain the whole-cipher correlation spectrum from it. Unlike the methods common in the literature, we do not compose round-by-round estimates but perform a precise computation of the fixed-key Walsh spectrum. We report the first exact DL distinguishers for KATAN-32, BEANIE, KeeLoq, and RC5-16. Their correlations are measured over the complete codebook and confirmed on independently sampled keys, rather than estimated by composing round-by-round trails. For RC5-16, this black-box approach also avoids modelling the data-dependent rotations that complicate trail searches based on mixed-integer linear programming (MILP) or Boolean satisfiability (SAT).

[1]The source code of the library is available at https://github.com/hadipourh/fwht.

For KeeLoq, the exact analysis also exposed ciphertext-side inputs to the nonlinear feedback function that refine Bogdanov's basic linear slide attack [Bog07] from $2^{50.6}$ to approximately $2^{48.82}$ encryptions under the source's cost model, without changing its data or memory requirements. For SIMON-32 we improve the best published correlations at several matched round counts, for example from $2^{-10.84}$ to $2^{-8.95}$ over 12 rounds, where the distinguisher we report also varies by 0.59 bits across keys against 13.8 bits for the published one. Because every candidate is evaluated exactly instead of being scored through a dominant trail, the ranking criterion becomes a parameter of the search. We rank by average strength, by stability across keys, or by strength on the weakest keys, and we state which criterion found each result.

- We analyze how the measured correlations depend on the key, rather than reporting a single value for each distinguisher. We use the geometric approach to cryptanalysis, which expresses a fixed-key correlation as a sum of terms whose signs depend on the round keys. Our procedure combines trail search, exact measurement of the predicted key dependence, and confirmation on independent keys under the cipher's key schedule. Where the dependence concentrates on a few parities of round-key bits, the procedure identifies these parities and confirms the resulting key classes. The procedure also detects cancellation among enumerated leading key-dependent terms and measures how much variation the tested candidate family leaves unexplained. The analysis shows that correlations across keys need not follow a normal distribution and that a root-mean-square value can be dominated by strong keys. For one of our distinguishers, a quarter of the keys belong to a weak class. Its root-mean-square correlation implies a data requirement about 40 times that estimated from the overall root-mean-square correlation. We therefore report how the required data changes with the fraction of keys an attack aims to cover.

Paper Outline.

The paper is organized as follows. Section 2 fixes notation and proves the fast WHT and its involution property. Section 3 describes the reusable `LibFWHT` implementation methodology. Section 4 reports implementation measurements. Section 5 provides a consolidated comparison with existing tools. Section 6 presents the cryptanalytic context and the full-codebook application to cryptanalysis. Section 7 instantiates the method for the cipher targets. Section 8 analyzes how the measured correlations depend on the key and what that dependence means for reporting a distinguisher. Section 9 concludes.

2 Preliminaries

2.1 Boolean Function Theory

We write $\mathbb{F}_2 = \{0, 1\}$ as the field with two elements and denote $\mathbb{F}_2^n$ as the n-dimensional vector space over $\mathbb{F}_2$, the elements of which we often identify as n-bit strings. Our convention follows standard treatments of Boolean Fourier analysis and cryptographic Boolean functions as in [O'D14, Car21]. For $\alpha, x \in \mathbb{F}_2^n$, the binary dot product is defined as

$$\alpha \cdot x = \sum_{i=0}^{n-1} \alpha_i x_i \in \mathbb{F}_2.$$

For a function value table stored in an array of length 2^n, we use natural binary order: the value at $x = (x_0, \ldots, x_{n-1})$ is stored at offset $\sum_{i=0}^{n-1} x_i 2^i$, so x_0 is the least-significant bit.

Algorithm 1: Fast Walsh-Hadamard Transform

Input: List $\mathcal{L}$ of length $N = 2^n$, containing $\mathcal{L}[x] = f(x)$.
Output: The same list, overwritten by $\mathcal{L}[\alpha] = \widehat{f}(\alpha)$.

```
for k ← 0 to n − 1 do
    ℓ ← 2^k;
    for b ← 0 to N/(2ℓ) − 1 do
        for j = 0, 1, ..., ℓ − 1 do
            u ← L[2ℓb + j];
            v ← L[2ℓb + j + ℓ];
            L[2ℓb + j] ← u + v;
            L[2ℓb + j + ℓ] ← u − v;
```

Definition 1 (Walsh-Hadamard transform)**.** Let $f : \mathbb{F}_2^n \to \mathbb{R}$ be a real-valued function. The *Walsh-Hadamard transform* of f is the function $\widehat{f} : \mathbb{F}_2^n \to \mathbb{R}$ defined by

$$\widehat{f}(\alpha) = \sum_{x \in \mathbb{F}_2^n} f(x)(-1)^{\alpha \cdot x}.$$

Definition 2 (Walsh spectrum of a Boolean function)**.** Let $g : \mathbb{F}_2^n \to \mathbb{F}_2$ be a Boolean function and define its sign encoding by

$$\chi_g(x) = (-1)^{g(x)} \in \{-1, +1\}.$$

Its *Walsh spectrum* is the WHT of χ_g, namely

$$W_g(\alpha) = \widehat{\chi_g}(\alpha) = \sum_{x \in \mathbb{F}_2^n} (-1)^{g(x)+\alpha \cdot x}.$$

Viewing g as a real-valued 0/1 function, its WHT $\widehat{g}$ is not generally W_g. With $N = 2^n$, the two are related by

$$\widehat{g}(\alpha) = \frac{N\,\delta_{\alpha,0} - W_g(\alpha)}{2},$$

where $\delta_{\alpha,0}$ is the Kronecker delta.

For $n \in \mathbb{N}$, the Hadamard-Sylvester matrix H_n is the $2^n \times 2^n$ matrix with entries

$$(H_n)_{\alpha,x} = (-1)^{\alpha \cdot x}.$$

It satisfies

$$H_n = H_1^{\otimes n}, \qquad \text{where} \qquad H_1 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \tag{1}$$

This Kronecker factorization is the key property underlying the butterfly structure in the fast transform. It also explains why the transform suits hardware built to multiply small dense matrices, which Subsection 3.6 returns to.

2.2 Fast Walsh-Hadamard transform

Theorem 1 (Fast Walsh-Hadamard transform)**.** *Let $f : \mathbb{F}_2^n \to \mathbb{R}$ be stored in a list $\mathcal{L}$ of length $N = 2^n$. Algorithm 1 computes the Walsh-Hadamard transform of f in-place using $\mathcal{O}(n2^n)$ arithmetic operations and $\mathcal{O}(2^n)$ storage.*

Proof. In pass k, the algorithm sets $\ell = 2^k$. It updates only pairs of indices that differ in bit k. Each update applies H_1 to one pair. Thus, pass k applies $I_{2^{n-k-1}} \otimes H_1 \otimes I_{2^k}$ to the state represented by $\mathcal{L}$. The algorithm executes these passes for $k = 0, 1, \ldots, n-1$ and applies the product

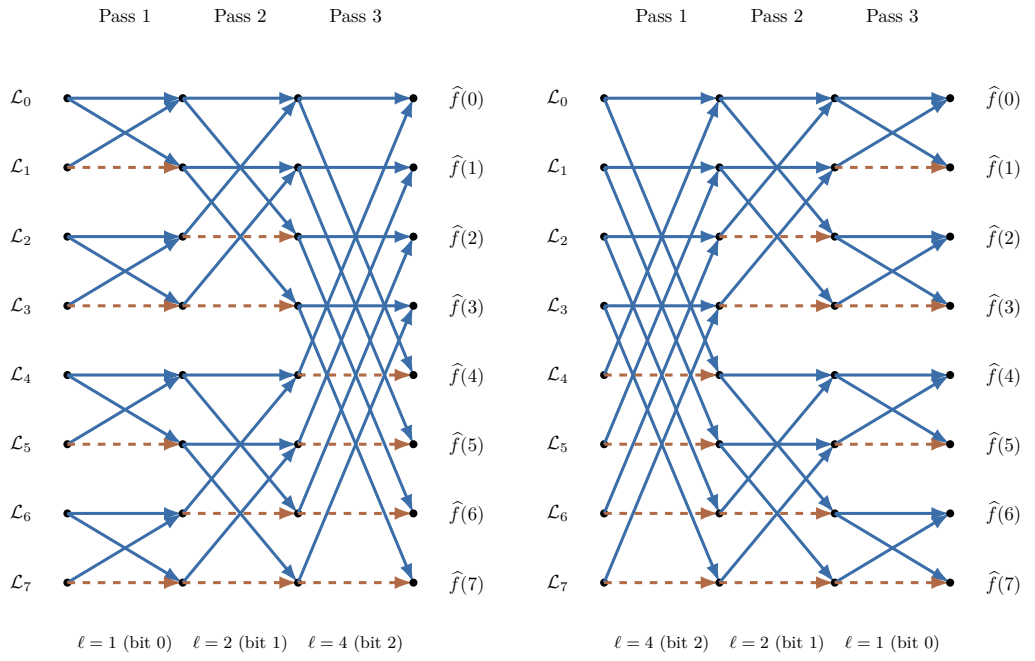
$$\prod_{k=0}^{n-1} (I_{2^{n-k-1}} \otimes H_1 \otimes I_{2^k}) = H_1^{\otimes n} = H_n.$$

Each pass performs $N/2$ butterflies, while each butterfly performs one addition and one subtraction, giving a total arithmetic cost of $nN = \mathcal{O}(n2^n)$. The transform overwrites the input and uses only constant auxiliary storage beyond the list. □

Lemma 1 (The passes commute)**.** *Let $P_k = I_{2^{n-k-1}} \otimes H_1 \otimes I_{2^k}$ be the pass of Algorithm 1 that updates the index pairs differing in bit k. Then $P_j P_k = P_k P_j$ for all j, k, so every ordering of the n passes computes H_n.*

Proof. Identify $\mathbb{R}^{2^n}$ with $\bigotimes_{i=0}^{n-1} \mathbb{R}^2$, one factor per bit position. The pass P_k acts as H_1 on the factor of bit k and as the identity on every other factor. For $j \neq k$, the two passes therefore act nontrivially on disjoint factors, and operators on disjoint tensor factors commute, since $(A \otimes I)(I \otimes B) = A \otimes B = (I \otimes B)(A \otimes I)$. The product of the n passes is thus independent of their order, and Theorem 1 evaluates it as $H_1^{\otimes n} = H_n$. □

An implementation may therefore choose the pass order freely, which Figure 1 illustrates for $n = 3$. The freedom is not only cosmetic: a pass whose stride keeps both elements of every pair inside one thread, one warp, or one cache line costs less than one that does not, so an implementation can group the cheap passes together and pay for the expensive ones once.



(a) Ascending stride (b) Descending stride

Figure 1: Two orders of the three passes for the 8-point Walsh-Hadamard transform, both computing $H_1^{\otimes 3} = H_3$. Each pass applies H_1 to $N/2$ independent element pairs, solid arrows denote addition and dashed arrows subtraction, and Lemma 1 explains why the order is free.

Lemma 2 (Walsh-Hadamard involution)**.** *For every $f\colon \mathbb{F}_2^n \to \mathbb{R}$, the Walsh-Hadamard transform satisfies*

$$\hat{\hat{f}}(x) = 2^n f(x).$$

Equivalently, H_n fulfills $H_n^2 = 2^n I_{2^n}$.

Proof. For $x, y \in \mathbb{F}_2^n$, character orthogonality gives

$$\sum_{\alpha \in \mathbb{F}_2^n} (-1)^{\alpha \cdot x + \alpha \cdot y} = \begin{cases} 2^n, & x = y, \\ 0, & x \neq y. \end{cases}$$

Expanding the transform twice and interchanging the finite sums yields

$$\hat{\hat{f}}(x) = \sum_{y \in \mathbb{F}_2^n} f(y) \sum_{\alpha \in \mathbb{F}_2^n} (-1)^{\alpha \cdot x + \alpha \cdot y} = 2^n f(x).$$

□

A second FWHT followed by division by 2^n therefore recovers the original array. `LibFWHT` uses this identity to implement inverse transforms without separate kernels.

3 `LibFWHT`: Architectural Framework

`LibFWHT` is an open-source C99 realization of dense fast Walsh–Hadamard transforms (FWHTs), developed specifically for cryptanalytic workloads rather than for the optimization of a single transform benchmark. Its formulation is guided by three principles. First, the default transform is unnormalized, allowing the same computational kernels to represent exact integer spectra as well as floating-point correlations. Second, for sufficiently large transform sizes, dense FWHT computation is governed primarily by memory traffic rather than by the arithmetic cost of an individual butterfly; consequently, locality and data movement are central to the computational model. Third, the external application programming interface (API) is deliberately compact, permitting the spectral primitive to be incorporated into exhaustive-search procedures without requiring backend-specific glue code.

3.1 Transform Semantics, Data Representations, and Numerical Conventions

The computational interface is defined to follow the mathematical structure of the Walsh–Hadamard transform directly. Transform sizes are specified by the physical array length $N = 2^n$, rather than by the Boolean dimension n. Each operation returns a status code that distinguishes invalid transform lengths, unavailable computational backends, and integer-overflow conditions.

The numerical representation includes in-place `int8`, `int32`, and `double` transforms. An additional checked `int32` realization detects arithmetic overflow. Contiguous-batch, packed-Boolean, and explicit-backend variants provide the representations required for key sweeps, S-box analysis, and controlled experiments. CUDA batch transformations additionally admit `fp32` and `fp16` data. An S-box is treated here as a vectorial Boolean function represented by a substitution table, while a linear approximation table (LAT) is formed from the Walsh spectra of its component Boolean functions.

A single sign convention governs every correlation reported in this paper. The numerical transform operates directly on the supplied values; consequently, a Boolean function must be represented using $0 \mapsto +1$ and $1 \mapsto -1$.

For example, consider the nonlinear function $g(x) = x_0x_1 \oplus x_2$ over $\mathbb{F}_2^3$, with x_0 varying fastest. Its raw truth table is $[0,0,0,1,1,1,1,0]$, and its corresponding signed representation is $[1,1,1,-1,-1,-1,-1,1]$. Applying the transform in Listing 1 produces $[0,0,0,0,4,4,4,-4]$. Thus $W_g(4) = W_g(5) = W_g(6) = 4$ and $W_g(7) = -4$, with the other coefficients equal to zero, in agreement with Definition 2.

```
int32_t spectrum[8] = {1, 1, 1, -1, -1, -1, -1, 1};
fwht_status_t status = fwht_i32(spectrum, 8);
/* spectrum is now [0, 0, 0, 0, 4, 4, 4, -4] */
```

Listing 1: The sign encoding and spectrum of $g(x) = x_0x_1 \oplus x_2$ on $\mathbb{F}_2^3$.

When a truth table is represented in packed-bit form, `fwht_boolean_packed` performs the same sign conversion internally and returns the unnormalized `int32` Walsh spectrum without explicitly materializing an intermediate ± 1 array. The `pyfwht` package exposes equivalent transforms over NumPy arrays and preserves the same numerical semantics.

3.2 Unified Computational Model and Backend Abstraction

The transform is exposed through a C API, Python bindings, and a command-line front end, all of which correspond to the same underlying mathematical operation. Following validation, representation conversion, and packing, a common dispatch layer selects the computational realization. Regardless of the selected backend, the resulting coefficients obey the same unnormalized Walsh–Hadamard convention.

In the terminology used throughout this section, CPU execution denotes computation on the host processor, single-instruction multiple-data (SIMD) denotes vector execution within an individual processor core, and CUDA denotes NVIDIA's GPU programming platform. These execution models differ only in the architectural mapping of the butterfly network; they do not modify the mathematical transform.

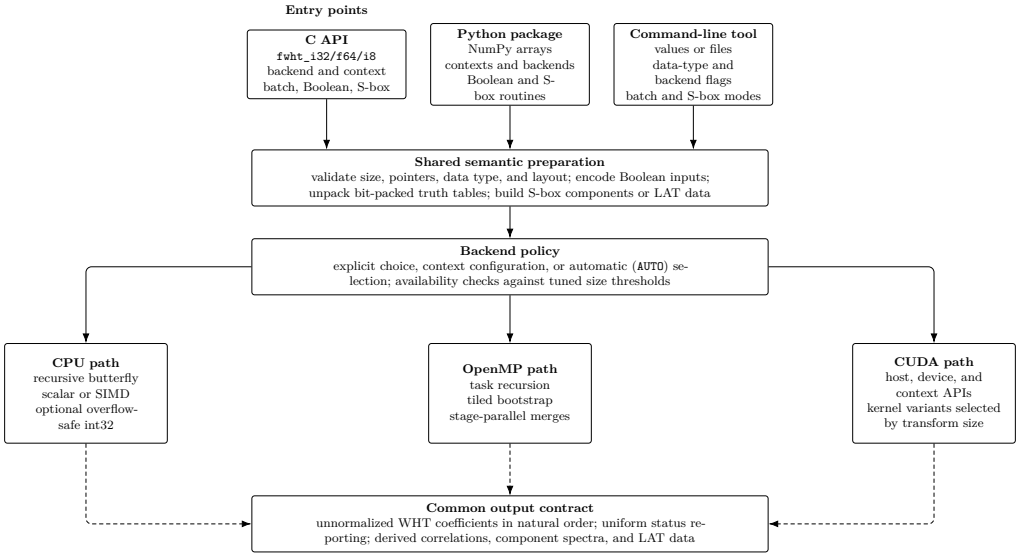


Figure 2: Unified computational organization of `LibFWHT`. Backend-specific realizations modify the architectural mapping of the butterfly network while preserving the unnormalized transform convention, natural coefficient ordering, and common numerical semantics.

Figure 2 therefore describes a common semantic and computational contract rather than a collection of implementation files. The interfaces differ in their representations and

modes of interaction, whereas the backends differ in how the same butterfly computation is mapped onto the underlying hardware. Across all paths, the coefficient convention and error-reporting discipline remain invariant.

3.3 Computational Complexity and Memory Bandwidth

The dense FWHT is fundamentally a memory-bound computation at sufficiently large problem sizes. Let $N = 2^n$. The transform consists of n passes, each containing $N/2$ butterflies. Every butterfly requires a constant number of arithmetic operations and transfers a constant number of machine words. Consequently, both the total arithmetic work and the total word traffic scale as $\Theta(N \log_2 N)$.

If a machine word occupies w bytes, the arithmetic intensity of the FWHT can be expressed as

$$I_{\mathrm{FWHT}} = \frac{c_1 N \log_2 N}{c_2 w N \log_2 N} = \Theta(1),$$

where c_1 and c_2 are machine-dependent constants.

The arithmetic intensity therefore does not increase with N. Once the active transform no longer resides comfortably within cache or shared memory, runtime becomes increasingly determined by memory bandwidth and locality rather than arithmetic throughput. This property determines the architectural mapping used throughout `LibFWHT`. SIMD execution is most consequential for small transforms that remain cache-resident, whereas OpenMP tiling, CUDA memory coalescing, and persistent GPU contexts become increasingly important for large transforms.

3.4 Vectorized Mapping on Scalar–SIMD Processor Architectures

The CPU realization consists of a scalar computational kernel that provides the portable baseline together with explicit vector intrinsics when the processor exposes an appropriate vector instruction-set architecture (ISA). At initialization, the execution environment identifies the available vector ISA through CPUID on x86 processors or through compile-time feature flags on ARM systems. The resulting dispatch selects among AVX2, SSE2, and NEON.

Explicit intrinsics are used because the regular structure of the Walsh–Hadamard butterfly permits direct control over the vectorized arithmetic and data movement. The number of simultaneously processed coefficients depends on both the ISA and numerical datatype. AVX2 processes eight `int32` lanes or four `double` lanes per vector instruction. SSE2 and NEON process four `int32` lanes or two `double` lanes.

For the AVX2 `int32` realization, eight butterfly pairs within a transform pass are evaluated in one vectorized step. Both halves of the corresponding pairs are loaded, after which the vector sum is written to one half and the vector difference to the other.

For batched spectra, SIMD parallelism may alternatively be applied across independent transforms. This mapping is advantageous when the size of an individual spectrum is insufficient to occupy all available vector lanes.

3.5 Hierarchical Parallel Decomposition on Shared-Memory Multiprocessors

A direct stage-parallel mapping assigns all available threads to every transform pass. Under this organization, each thread repeatedly traverses the complete N-element array. Such a schedule induces repeated cache-line replacement and provides weak temporal reuse.

The multi-core realization instead exploits the recursive block structure of the Hadamard matrix,

$$H_n = \begin{bmatrix} H_{n-1} & H_{n-1} \\ H_{n-1} & -H_{n-1} \end{bmatrix}.$$

At moderate transform sizes, this identity permits the computation to be decomposed recursively into two contiguous $N/2$-point subproblems. Sibling computational tasks consequently operate on disjoint cache lines, preserving both locality and non-uniform memory access (NUMA) first-touch placement [DM98, Ope21].

For very large transforms, `LibFWHT` first evaluates cache-resident bootstrap tiles and then performs only the upper transform passes using stage-parallel loops. The bootstrap tile consists of 2^{17} double-precision values, corresponding to 1 MiB, and therefore fits comfortably within the L2 cache of the systems evaluated in this work.

For T available processor threads, the recursive task depth is $\lfloor \log_2 T \rfloor + 2$. This construction produces between two and four leaf tasks per thread. The resulting decomposition provides sufficient parallel work while avoiding the creation of an excessive number of fine-grained runtime tasks.

In `AUTO` mode, sufficiently small transforms remain assigned to the single-thread SIMD realization because thread initialization and synchronization costs would otherwise dominate computation. If the OpenMP backend is explicitly selected, however, the transform is executed using OpenMP even at small sizes. This behavior permits controlled benchmarking of the different parallel mappings.

3.6 GPU Mapping of the Walsh–Hadamard Butterfly Network

The CUDA realization maps the Walsh–Hadamard butterfly network onto NVIDIA GPU architectures [NBGS08]. Two computational regimes arise depending on whether the complete active working set can reside in on-chip shared memory.

When the full transform fits within shared memory, a single thread block loads the input coefficients, evaluates all n transform passes on chip, and then writes the result once to global memory. Synchronization between passes is enforced using `__syncthreads()`.

For transform lengths exceeding this regime, `LibFWHT` executes one CUDA kernel per transform pass and maps one butterfly to each GPU thread. This formulation maintains simple index expressions and exposes the memory-access structure directly.

The early transform passes are the least efficient with respect to the GPU global-memory subsystem. For butterfly stride $\ell < 32$, consecutive threads within the same warp access strided memory locations, leading to poor global-memory coalescing. Once $\ell \geq 32$, the referenced addresses become substantially closer to consecutive locations, and the later passes use memory bandwidth more effectively. Consequently, the runtime of sufficiently large CUDA transforms follows device-memory bandwidth considerably more closely than raw GPU arithmetic throughput.

Persistent device contexts provide a complementary mechanism for repeated spectral evaluations by avoiding repeated device-memory allocation and deallocation between transform invocations. A device-pointer API additionally permits input and output arrays to remain GPU-resident when the Walsh–Hadamard transform is composed with another device kernel. DLPack zero-copy exchange [DML24] extends the same device-residency principle to Python-based computational workflows.

The Kronecker structure in Equation 1 yields an alternative architectural mapping. Since $H_n = H_1^{\otimes n}$, any sequence of m consecutive transform passes is equivalent to applying H_m, a dense $2^m \times 2^m$ matrix, independently to blocks containing 2^m coefficients.

For $m = 4$, the corresponding computation becomes a 16×16 matrix operation. This dimensionality matches the matrix operations that recent NVIDIA GPU matrix units, commonly termed Tensor Cores, can evaluate using a single instruction [MCL+18]. Under

this formulation, one such matrix instruction replaces four conventional transform passes over memory with computation whose intermediate values remain in registers.

`LibFWHT` provides this Tensor-Core-based mapping for `fp16` data on GPU compute capabilities 8.0 through 9.x and for transform lengths ranging from 2^8 to 2^{15}. The realization adapts the kernels of `HadaCore` [Met24].

The numerical exactness of this mapping, however, is restricted to a comparatively narrow range, and it is therefore excluded from the exact cryptanalytic analysis reported in this paper. A half-precision floating-point value contains an 11-bit significand. It consequently represents every integer exactly through 2^{11}, and every even integer exactly through 2^{12}.

For a ± 1-valued input, every intermediate and final Walsh–Hadamard value is an integer, has the same parity as N, and has absolute magnitude no greater than N. The `fp16` result is therefore guaranteed to remain exact only when $N \leq 2^{12}$.

The supported `fp16` transform lengths nevertheless extend to 2^{15}. At this scale, the spacing between representable half-precision values grows to 16. A sufficiently large Walsh coefficient that is not an integer multiple of 16 is consequently rounded.

The same reasoning determines the exactness ranges of the other floating-point representations. An `fp32` value contains a 24-bit significand and is exact for $N \leq 2^{24}$, whereas `fp64` contains 53 significand bits and therefore remains exact for transform sizes far exceeding any feasible transform length considered here.

The principal problem with the `fp16` realization is not merely the magnitude of its numerical error, but the spectral locations at which such errors arise. For a random Boolean function, Walsh coefficients are typically of magnitude approximately $\sqrt{N}$, which is well within the exactly representable range. The half-precision transform therefore usually agrees with the exact result, but errors occur preferentially for inputs with unusually large Walsh coefficients. These inputs are precisely those of greatest cryptanalytic interest.

Rounding may additionally map an exact zero coefficient to a nonzero coefficient. It can also reverse the ordering of two candidates whose correlations differ only slightly, which is exactly the behavior that the ranking procedure in Subsection 6.3 must avoid. For this reason, all exact full-codebook results reported in this paper are obtained using only the integer and double-precision computational paths.

3.7 Batched Spectral Evaluation, Packed Boolean Representations, and Numerical Correctness

Cryptanalytic computations frequently require the evaluation of thousands of relatively small Walsh–Hadamard transforms rather than a single large transform. `LibFWHT` therefore defines batched computation on both CPU and GPU architectures.

On CPU systems, SIMD lanes may be populated using coefficients from independent transforms, thereby providing vector utilization even when an individual transform is too small to occupy the full vector width. On GPUs, persistent computational contexts amortize both `cudaMalloc`/`cudaFree` operations and host–device data-transfer overhead across extended sequences of transforms.

The packed-Boolean representation introduced in Subsection 3.1 is implemented by `fwht_boolean_packed`. Given the truth table of a Boolean function g, the representation is read as a sequence of `uint64_t` words. Individual Boolean values are converted to their corresponding ± 1 representation, and the resulting unnormalized integer Walsh spectrum is written to an `int32` output buffer. Normalized correlations are then obtained by dividing the Walsh coefficients by N.

For transform lengths no greater than 2^{16}, an explicitly selected GPU realization preserves the truth table in packed form during host-to-device transfer and performs its expansion on the GPU itself. A dedicated Boolean GPU context further reuses both the packed-input and signed-output device buffers across repeated transforms.

The `fwht_i32_safe` entry point detects overflow for arbitrary integer-valued input vectors. Compiler overflow intrinsics are used to determine whether any `int32` butterfly operation exceeds the representable integer range. In such a case, the transform returns `FWHT_ERROR_OVERFLOW` rather than silently applying integer wraparound. This checked computation is slower than the unchecked `int32` realization, but is necessary when no a priori bound is available for $N \max_i |x_i|$.

The computational formulation imposes several constraints. Transform lengths must be powers of two. The default Walsh–Hadamard output is unnormalized; consequently, division by N must be applied explicitly whenever normalized correlations or inverse-style recovery are required. Contiguous-batch representations assume that every transform has equal length and that the transforms occupy contiguous storage.

GPU computation is most effective for batched or device-resident workloads in which allocation and data-transfer costs can be amortized over multiple evaluations. The optional `fp16` realization is unsuitable for cryptanalytic analyses that require exact integer Walsh coefficients, for the numerical reasons established in Subsection 3.6. Different host threads may simultaneously process distinct data buffers, but concurrent access to a shared buffer requires external synchronization.

The purpose of this formulation is therefore not to abstract away the Walsh–Hadamard butterfly network itself. The butterfly remains the underlying spectral primitive. Instead, the framework removes the repeated auxiliary mechanisms required to deploy that primitive across cryptanalytic computations. Key-search loops, difference sweeps, and S-box processing pipelines typically vary more frequently than the spectral operation on which they depend.

4 Performance Evaluation of `LibFWHT`

The implementation experiments characterize the shared transform engine. They measure backend scaling on representative CPUs and GPUs and check that every backend preserves the same unnormalized transform convention.

4.1 Measurement Methodology

We measured an Apple M4 system (ARM64, NEON, 10 cores), a dual-socket AMD EPYC 7742 system (x86-64, AVX2, 128 cores), an NVIDIA H200 (143771 MiB HBM3e), and an NVIDIA RTX PRO 6000 Blackwell (97887 MiB GDDR7). The Apple M4 host ran macOS with 24 GiB unified memory. The EPYC host provided 256 logical processors across two non-uniform memory access nodes, and its OpenMP runs used all of them. CPU builds used C99, compiler optimization, native-code generation, and the platform OpenMP runtime. GPU builds used `nvcc` version 12.6.68 on the H200 and 12.8.93 on the RTX PRO 6000, both with optimization level O3. The H200 host ran driver 595.91.07 with two AMD EPYC 9554 processors, and the RTX PRO 6000 host driver 590.44.01 with two AMD EPYC 9534 processors. The latter device reports compute capability 12.0, and the build targets it natively rather than reaching it through just-in-time compilation. For the CPU and OpenMP results we report medians of 20 paired samples, balanced so that each backend runs first in half of them. For GPUs, we report means from 10 repetitions after one warm-up iteration with persistent contexts and pre-allocated memory. All timings refer to unnormalized dense transforms unless a caption states otherwise. Figure 3 uses a separate 20-sample latency sweep and reports medians.

4.2 Verification Strategy

We verified `LibFWHT` in three ways. First, Section 2 proves the butterfly algorithm and the involution property in Lemma 2. Second, scalar, SIMD, OpenMP, and CUDA paths produced identical outputs for common integer and Boolean inputs. Third, the valid `FFTW` discrete Hartley transform (DHT) route in Appendix A and the involution identity provided independent floating-point checks. Together, these checks validate the numerical convention used in the timing experiments.

4.3 CPU and OpenMP Performance

To determine when host threading pays off, we measured one moderate size and one large size on both hosts. Table 1 reports representative points from the larger sweep.

Table 1: CPU and OpenMP transform latency.

Platform	Backend	Size	Median (ms)
Apple M4	CPU	2^{24}	33.8
Apple M4	OpenMP	2^{24}	13.3
Apple M4	CPU	2^{30}	2698.1
Apple M4	OpenMP	2^{30}	1344.6
AMD EPYC 7742	CPU	2^{24}	126.0
AMD EPYC 7742	OpenMP	2^{24}	7.1
AMD EPYC 7742	CPU	2^{30}	9401.7
AMD EPYC 7742	OpenMP	2^{30}	953.8

OpenMP helps on both systems, and the size of the gain follows the worker count: about 2.53× with 10 workers on the Apple M4 against about 17.77× with 256 workers on the EPYC. On both systems the gain then falls at 2^{30}, to 2.01× and 9.86×. The gains stay below the worker counts, and they shrink as the transform grows, because dense FWHT has constant arithmetic intensity. Once the working set exceeds cache, memory bandwidth rather than arithmetic throughput limits scaling. At small sizes the same threading becomes a cost rather than a benefit: with 256 workers the EPYC OpenMP path is slower than one core below 2^{17}, which is why the automatic backend policy applies a size threshold before selecting it.

Figure 3 identifies the Apple M4 latency crossover. The 10-worker OpenMP path first overtakes the CPU path that uses one core at 2^{18}. It is 4.13× faster at 2^{22} and remains 2.01× faster at 2^{30}. The smaller gain at the large end is consistent with the memory-bandwidth limit.

The GPU curves separate device-resident and transfer-inclusive execution. Kernel latency remains below 1 ms through 2^{25} and reaches 17.709 ms at 2^{30}. At that size, host-to-device plus device-to-host transfer contributes 341.180 ms, producing a 358.880 ms CUDA-event end-to-end total. The transfer cost shows why GPU use is most effective for batched or device-resident cryptanalytic workloads.

4.4 GPU Performance

We measured large `int32_t` transforms on two graphics processors that differ in memory technology, an NVIDIA H200 with HBM3e and an NVIDIA RTX PRO 6000 Blackwell with GDDR7. The benchmark reports billions of arithmetic operations per second (GOps/s) using nN operations for a length-N transform, where $n = \log_2 N$. Each device is reported twice, once for the kernel alone on data that already resides on the device, and once for the transfer-inclusive path that also copies the array to the device and back. Every row of

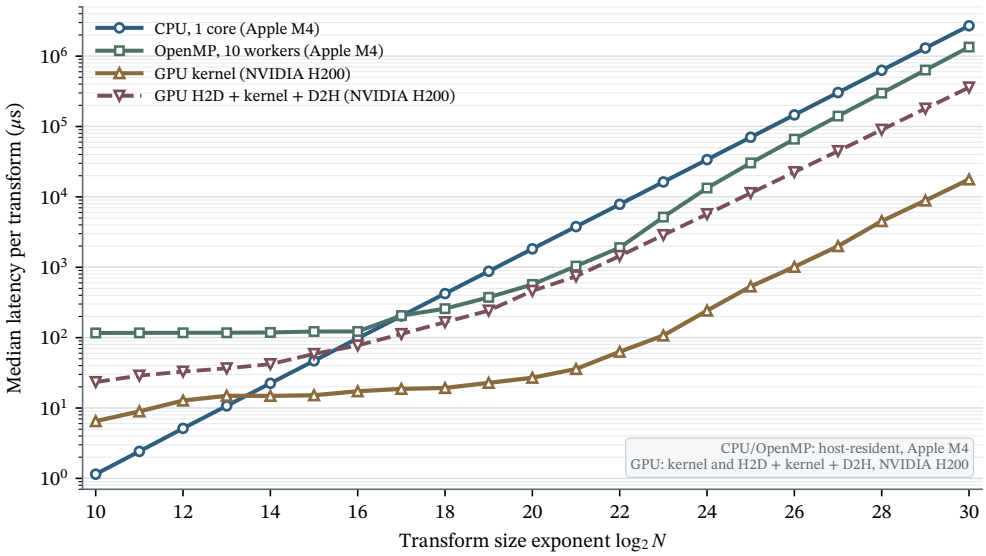


Figure 3: `LibFWHT` exposes distinct CPU, OpenMP, and GPU operating regimes. Each curve is the median of 20 samples. GPU allocation, host-side preparation, and validation are excluded, and the cross-platform curves are not a CPU–GPU speedup comparison.

Table 2: GPU timings for 2^{30} `int32_t` elements, reported as medians of 20 samples.

Platform	Memory	Path	Running time (ms)	Throughput (GOps/s)
NVIDIA H200	HBM3e	device-resident kernel	17.709	1818.93
NVIDIA H200	HBM3e	transfer-inclusive	358.880	89.76
RTX PRO 6000	GDDR7	device-resident kernel	41.327	779.45
RTX PRO 6000	GDDR7	transfer-inclusive	413.465	77.91

the recorded data carries the device name, driver version, and CUDA runtime version that produced it.

At 2^{30} the H200 kernel reaches 1818.93 GOps/s and the RTX PRO 6000 kernel reaches 779.45 GOps/s. Per-pass kernel launches, synchronization, and the less-coalesced early passes add cost on both devices. The transfer-inclusive rows fall to 89.76 and 77.91 GOps/s, because at this size the copies dominate the kernel by more than an order of magnitude. The two devices are close once transfers are included, and far apart without them.

The size of that gap decides how a caller should organize a search, so we state it as a rule rather than as a single measurement. Moving the array costs about 20 times what transforming it costs, so a workflow that copies results back after every transform runs roughly 20 times longer than one that leaves them on the device and copies only the final answer. This is what the device-pointer entry points, persistent contexts, and DLPack exchange of Subsection 3.6 are for: they let a sweep pay for the transfer once instead of once per transform. For a sweep that transforms once per candidate over millions of candidates, this ratio rather than the kernel speed sets the running time. The effect belongs to the workload rather than to any one data type, as the next measurement shows.

Table 3 compares the four data types the CUDA backend supports on a single batched shape, 10,000 transforms of length 2^{15}, timing the kernel alone and then the kernel followed by the copy back to the host. Three of the four behave the same way once element size

is taken into account. The `int32_t` and `fp32` kernels move the same number of bytes and take the same time, and the `fp64` kernel moves twice as many bytes and takes 1.82 times as long. That is the bandwidth limit of Subsection 3.3 once more, seen here across data types on one device rather than across devices. The `fp16` Tensor Core kernel is the exception, at 3.47 times faster than `fp32` while moving half the bytes. Halving the element size accounts for a factor of two of that, and the matrix units of Subsection 3.6 account for the remaining 1.73. The copy dominates every column, taking at least 94% of the transfer-inclusive time, and it takes the largest share from the fastest kernel.

Table 3: H200 timings for 10,000 transforms of length 2^{15}. Medians of 20 samples after 2 warm-up runs.

Data type	Kernel (µs)	Kernel + copy (ms)	Kernel (%)
`int32_t`	2304.6	44.7	5.2
`fp64`	4276.3	105.4	4.1
`fp32`	2349.7	52.9	4.4
`fp16` (Tensor Core)	677.0	26.0	2.6

The ordering between the two devices depends on the transform size, as Figure 4 shows. Below 2^{26} the RTX PRO 6000 kernel is the faster of the two, by up to 1.6× at 2^{24}, because the work still fits in cache and the comparison is then decided by launch overhead and arithmetic throughput. Its throughput peaks at 2^{24} and then falls by more than a factor of three. From 2^{26} upward the working set exceeds cache, the transform becomes bandwidth bound, and the H200 leads by a factor that settles near 2.3×. This is the same bandwidth limit that bounds the OpenMP gains in Subsection 4.3, observed here across two memory technologies rather than across core counts. A single size therefore cannot rank two graphics processors for this workload: the answer changes with the transform.

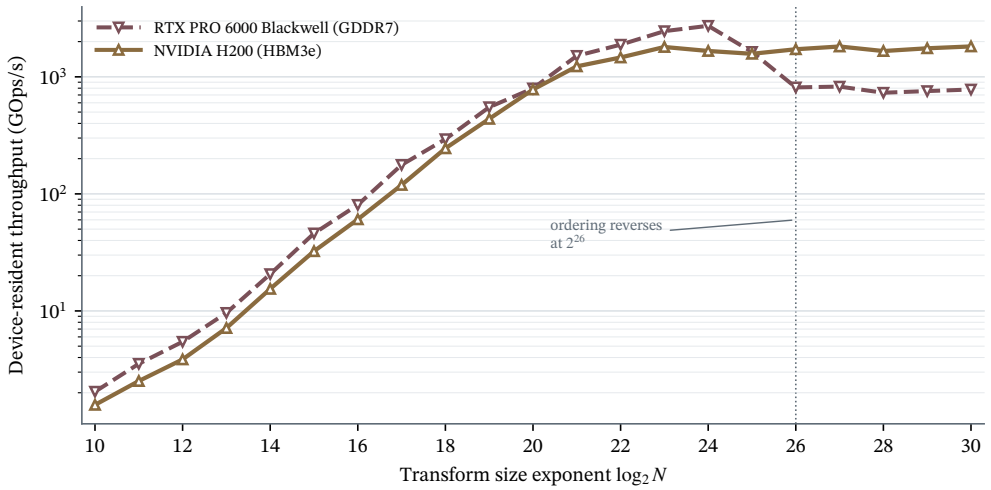


Figure 4: Memory technology decides which graphics processor is faster, and only above a threshold size. Both devices run the same device-resident `int32_t` sweep; throughput counts nN operations for a length-N transform. Each point is the median of 20 samples.

5 Comparison with Prior Work

This section compares LibFWHT with prior software under matching conditions. Table 4 separates the two direct comparisons from broader workflow context. The detailed timing tables and harness settings appear in Appendix A and Appendix B.

Table 4: Direct library comparisons under matched conditions, not a universal ranking.

Tool	Matching conditions	Measured result
FFTW	Apple M4; double-precision arrays; valid multidimensional size-two DHT route; warm plan reused and planning excluded.	LibFWHT is 2.5 to 3.3× faster with one thread, and 1.0 to 2.1× faster with 10 threads at 2^{20} and above.
sboxU	macOS; one CPU core; identical Boolean truth tables; unnormalized $0 \mapsto +1$, $1 \mapsto -1$ spectra; 20 balanced paired samples per size.	At 2^{29} entries, LibFWHT is 2.00× faster.

5.1 Dense Floating-Point Transforms

FFTW can compute a WHT through a multidimensional DHT with every dimension equal to two. A flat one-dimensional FFT does not compute a WHT because it uses characters of a different group. Appendix A derives the valid route and reports the full timing table. The matched single-thread comparison favors LibFWHT by 2.5 to 3.3× at every measured size, with the margin narrowing as the transform grows. With 10 threads, both libraries become bandwidth-bound, and LibFWHT retains a 1.0 to 2.1× advantage for transforms of 2^{20} and above. Below that size the threaded comparison is governed by thread startup rather than by transform speed, and neither library holds a reproducible lead. FFTW also provides an independent floating-point correctness baseline.

This comparison needs a word about its scope. FFTW has no native WHT, so the route used here is the best one available rather than one its authors set out to optimize. A transform of length 2^{30} becomes an array of rank 30 in which every dimension has size two, whereas FFTW is planned around a small number of dimensions of substantial size. The result should therefore be read as follows: LibFWHT is faster than the best route FFTW offers for this transform, which is not the same as being faster than FFTW on the transforms it was built for. That distinction matters for the claim, and it also describes the situation a reader meets in practice, because a tool that needs a WHT and reaches for FFTW obtains exactly this route. We measured it on terms favorable to FFTW: it plans with its tuning mode, the plan is warm and reused across iterations, planning time is excluded and reported separately, both libraries receive identically aligned buffers, and both are forced to the same thread count.

5.2 Boolean Walsh-Spectrum Transforms

The sboxU comparison measures a native Boolean-spectrum routine rather than generic dense-transform throughput. The benchmark invokes `cpp_walsh_transform` on a Boolean `cpp_S_box`. Both paths receive the same truth table and use the same unnormalized signed-spectrum convention. The harness checks exact coefficient equality before timing each size. Table 23 gives the detailed timings. The measurements on one CPU core favor LibFWHT at every measured size, by 1.11× at 2^{10} rising to 2.00× at 2^{29}.

A search does not compute one spectrum at a time, so we also measured the batched workload in which both libraries transform the same group of Boolean functions across several workers. That case is reported in Table 24. LibFWHT leads at 14 of the 18 measured

sizes, and at every size from 2^{19} upward, by 1.86× to 3.70×. `sboxU` leads between 2^{16} and 2^{18} by at most 1.20×. The sizes where `LibFWHT` leads are the ones a full-codebook search occupies, which makes the batched result the more relevant of the two for the analyses in Section 7. `sboxU` has no graphics-processor backend, so a CUDA timing is not a like-for-like speedup over it. It does, however, measure what the absent backend costs. On the same batched workload and the same host, the `LibFWHT` device-resident kernel at 2^{27} is 612× faster than the `sboxU` host measurement, and 27× faster once the host-to-device and device-to-host copies are included. Those figures describe the value of having a device path at all, not a faster kernel for the same work.

5.3 Complementary Workflow Context

Other comparisons address different workloads and should not be reduced to one ranking. The `pyfwht` package improves over SymPy reference workflows by three to four orders of magnitude, which measures the cost of a pure-Python reference rather than a competing implementation; Table 27 gives the timings. On Apple M4, `pyfwht` leads `Hadamard.jl` at 19 of 25 measured sizes, by two orders of magnitude at the small end where interpreter and call overhead dominate. `Hadamard.jl` leads from 2^{25} upward, once the working set leaves cache and both implementations become bandwidth bound, by 1.02× at that size and 1.61× at 2^{30}. Table 25 reports that comparison. On x86, `FFHT` remains faster for its narrower single-thread assembly-oriented workload, leading by 3.12× to 6.11× once the data types are matched (Table 26). These results show a trade-off between specialization and the unified integer, floating-point, Boolean, CPU, and graphics-processor paths exposed by `LibFWHT`.

6 Application to Cryptanalysis

6.1 Cryptanalytic Context

The Walsh-Hadamard transform has been a central tool in symmetric-key cryptanalysis for over three decades. Early descriptions of Walsh-Hadamard transforms were given by Good, Shanks, and Fino and Algazi [Goo58, Sha69, FA76]. We explain how modern cryptanalysis uses the transform and define the DL quantity that `LibFWHT` evaluates.

6.1.1 Linear cryptanalysis and Walsh spectra

Linear cryptanalysis, introduced by Matsui [Mat93], identifies affine approximations of block ciphers. We write $\mathrm{Cr}_k^{\mathrm{Lin}}$ for linear correlation and $\mathrm{Cr}_k^{\mathrm{DL}}$ for DL correlation. The superscript identifies the analysis type, and the subscript identifies the fixed key. For a key-dependent permutation $E_k\colon \mathbb{F}_2^n \to \mathbb{F}_2^n$, the *linear correlation* for input mask α and output mask β is

$$\mathrm{Cr}_k^{\mathrm{Lin}}(\alpha,\beta) = 2^{-n} \sum_{x\in\mathbb{F}_2^n} (-1)^{\alpha\cdot x+\beta\cdot E_k(x)}. \tag{2}$$

This is the normalized entry of the LAT. Fix β and sign-encode $f_\beta(x) = \beta \cdot E_k(x)$. The unnormalized WHT of $(-1)^{f_\beta}$ over all x gives the full LAT column for β at once. One call to `fwht_boolean_packed` or `fwht_i32` computes one full LAT column in $\mathcal{O}(n2^n)$ time. Matsui's half-correlation convention is one half of correlation [Mat93]; `LibFWHT` reports correlations and leaves the factor of two to the user.

6.1.2 Differential cryptanalysis and difference distributions

Differential cryptanalysis, introduced by Biham and Shamir [BS90], tracks how input differences propagate to output differences under a fixed key. For a fixed key k and input

difference δ, the *difference distribution table* (DDT) counts output differences. Its entry at γ counts plaintext pairs $(x, x+\delta)$ satisfying $E_k(x) + E_k(x+\delta) = \gamma$. The DDT and LAT interact through the DL connectivity table, connecting differential and linear phenomena through the Walsh-Hadamard transform.

6.1.3 DL cryptanalysis and the DLCT

DL cryptanalysis, introduced by Langford and Hellman [LH94], combines a differential trail through the first part of a cipher with a linear trail through the second part. For a fixed key k and input difference δ, let

$$N_{k,\delta}(\gamma) = |\{x \in \mathbb{F}_2^n : E_k(x) + E_k(x+\delta) = \gamma\}|$$

be the histogram of output differences. For an output mask β, the *full-codebook DL correlation* is

$$\begin{aligned} \mathrm{Cr}_k^{\mathrm{DL}}(\delta, \beta) &= 2^{-n} \sum_{\gamma \in \mathbb{F}_2^n} N_{k,\delta}(\gamma)(-1)^{\beta \cdot \gamma} \\ &= 2^{-n} \sum_{x \in \mathbb{F}_2^n} (-1)^{\beta \cdot E_k(x) + E_k(x+\delta)}. \end{aligned} \tag{3}$$

The vector $(\mathrm{Cr}_k^{\mathrm{DL}}(\delta, \beta))_\beta$ is the normalized WHT of $N_{k,\delta}$. One call to `fwht_i32`, followed by division by 2^n, gives every output-mask correlation for the fixed pair (k, δ). When only one DL coefficient is needed, an implementation can use the symmetry between x and $x + \delta$ to avoid redundant work. This optimization does not change the definition in Equation 3. Bar-On, Dunkelman, Keller, and Weizman systematized this view in the DLCT [BDKW19], which tabulates correlations indexed by input differences and output masks. Hadipour, Derbez, and Eichlseder [HDE24] generalized the DLCT with a signed construction; the normalized convention in Equation 3 matches theirs. Our experiments in Section 7 compute $\mathrm{Cr}_k^{\mathrm{DL}}(\delta, \beta)$ directly through this formula.

We next analyze the reusable full-codebook method for block ciphers with input and output sizes up to 32 bits. For a fixed key, the method evaluates every plaintext. The resulting sign table or derivative histogram yields every output-mask correlation after one WHT. The method measures whole-cipher behavior directly instead of composing estimates from smaller cipher parts. Before describing how candidates are found and confirmed, we establish the ideal random-permutation baseline used to interpret the measured root-mean-square (RMS) values.

6.2 Random-Permutation Reference

We interpret measured RMS values with an ideal random-permutation reference. The reference treats the cipher as a uniformly random permutation on $\mathbb{F}_2^n$. It is a comparison scale, not a model of the reduced-round cipher instances under study.

Theorem 2 (Linear correlation of a random permutation)**.** *Let $N = 2^n \geq 4$, let P be a uniformly random permutation of $\mathbb{F}_2^n$, and fix non-zero $\alpha, \beta \in \mathbb{F}_2^n$. Put $M = N/2$ and*

$$L = \mathrm{Cr}_P^{\mathrm{Lin}}(\alpha, \beta) = \frac{1}{N} \sum_{x \in \mathbb{F}_2^n} (-1)^{\alpha \cdot x + \beta \cdot P(x)}.$$

Then L has the exact distribution

$$\Pr\left[L = \frac{4j - N}{N}\right] = \frac{\binom{M}{j}^2}{\binom{N}{M}}, \qquad 0 \leq j \leq M. \tag{4}$$

Consequently,

$$\mathbb{E}[L] = 0, \qquad \mathbb{E}[L^2] = \frac{1}{N-1},$$

and, as N grows, L is well approximated by a centered normal random variable with variance $1/(N-1)$.

Proof. Let $\chi_\gamma(z) = (-1)^{\gamma \cdot z}$. The functions χ_α and χ_β are balanced, so each takes $+1$ on M points and -1 on M points.

Let $A = \{x : \chi_\alpha(x) = +1\}$ and $B = \{y : \chi_\beta(y) = +1\}$. The random variable

$$J = |P(A) \cap B|$$

has the hypergeometric distribution

$$\Pr[J = j] = \frac{\binom{M}{j}\binom{M}{M-j}}{\binom{N}{M}} = \frac{\binom{M}{j}^2}{\binom{N}{M}}.$$

If $J = j$, then j points in A and j points in A^c contribute $+1$, while $M - j$ points in each half contribute -1. Thus the signed sum is $4j - N$, proving Equation 4. The hypergeometric mean and variance give

$$\mathbb{E}[J] = \frac{M}{2}, \qquad \mathrm{Var}(J) = \frac{M^2}{4(N-1)}.$$

Since $L = (4J - N)/N$, we get $\mathbb{E}[L] = 0$ and $\mathbb{E}[L^2] = 1/(N-1)$. The normal approximation follows from the standard central limit behavior of the hypergeometric law. □

Theorem 3 (DL correlation of a random permutation)**.** *Let $N = 2^n \geq 4$, let P be a uniformly random permutation of $\mathbb{F}_2^n$, and fix non-zero $\delta, \beta \in \mathbb{F}_2^n$. Put $M = N/2$ and*

$$D = \mathrm{Cr}_P^{\mathrm{DL}}(\delta, \beta) = \frac{1}{N} \sum_{x \in \mathbb{F}_2^n} (-1)^{\beta \cdot P(x) + P(x+\delta)}.$$

Although the sum defining D has N oriented inputs, the terms for x and $x + \delta$ are identical, so it averages M distinct unordered derivative pairs. For even s, let

$$\mathrm{pm}(s) = \frac{s!}{2^{s/2}(s/2)!}$$

be the number of perfect matchings on s labelled points. Then D has the exact distribution

$$\Pr\left[D = 1 - \frac{2t}{M}\right] = \frac{\binom{M}{t}^2 t!\, \mathrm{pm}(M-t)^2}{\mathrm{pm}(N)}, \tag{5}$$

for $0 \leq t \leq M$ with $M - t$ even, and the probability is 0 otherwise. Moreover,

$$\mathbb{E}[D] = -\frac{1}{N-1}, \qquad \mathbb{E}[D^2] = \frac{2N-5}{(N-1)(N-3)}.$$

As N grows, D is well approximated by a normal random variable with mean $-1/(N-1)$ and variance

$$\frac{2(N-2)^2}{(N-1)^2(N-3)} = \frac{2}{N} + O(N^{-2}).$$

Proof. Let $\chi_\beta(y) = (-1)^{\beta \cdot y}$. The character χ_β is balanced, so it takes each value on M output points.

The non-zero difference δ partitions the domain into M unordered pairs $\{x, x+\delta\}$. Their images under P form a uniformly random perfect matching on the N output points. Call an output point positive if $\chi_\beta(y) = +1$ and negative otherwise; there are M of each. Index the image pairs as (Y_i, Y_i') and put

$$Z_i = \chi_\beta(Y_i)\chi_\beta(Y_i') \in \{-1, +1\}.$$

Let T be the number of mixed-sign pairs in this random matching. Each mixed pair has $Z_i = -1$, and each same-sign pair has $Z_i = +1$. Because the full-codebook definition counts each domain pair twice,

$$D = \frac{1}{M}\sum_{i=1}^{M} Z_i = 1 - \frac{2T}{M}.$$

To count matchings with $T = t$, choose the t positive and t negative vertices that participate in mixed pairs, match them bijectively in $t!$ ways, and then match the remaining $M - t$ positive vertices and $M - t$ negative vertices internally. This is possible only when $M - t$ is even, giving the numerator in Equation 5; division by the total number $\mathrm{pm}(N)$ of perfect matchings proves the exact law.

The moments are easiest to read from pair signs. For a random matched pair (Y, Y'), sampled without replacement from the output set,

$$\mathbb{E}[\chi_\beta(Y)\chi_\beta(Y')] = \frac{(M-1) - M}{N-1} = -\frac{1}{N-1}.$$

Hence $\mathbb{E}[D] = -1/(N-1)$. For two distinct matched pairs, the four involved output points are distinct. If r of these four points are negative, the product of the two pair signs is $(-1)^r$, and therefore

$$\mathbb{E}[Z_i Z_j] = \frac{\sum_{r=0}^{4}(-1)^r \binom{M}{r}\binom{M}{4-r}}{\binom{N}{4}} = \frac{\binom{M}{2}}{\binom{N}{4}} = \frac{3}{(N-1)(N-3)}.$$

The middle equality is the coefficient of u^4 in $(1-u)^M(1+u)^M = (1-u^2)^M$. With $Z_i^2 = 1$, this gives

$$\mathbb{E}[D^2] = \frac{1}{M^2}\left(M + M(M-1)\frac{3}{(N-1)(N-3)}\right) = \frac{2N-5}{(N-1)(N-3)}.$$

Subtracting the squared mean gives the stated variance. The normal approximation follows from the central limit behavior of the number of mixed edges in a random perfect matching. □

The plotted absolute values $|L|$ and $|D|$ are approximated by the corresponding folded-normal laws. For $n = 32$, the DL mean is negligible, so its reference curve is visually indistinguishable from a half-normal curve matched to the DL second moment.

The resulting root-second-moment references are

$$\mathrm{RMS}_{\mathrm{random}}^{\mathrm{Lin}} = \frac{1}{\sqrt{N-1}}, \qquad \mathrm{RMS}_{\mathrm{random}}^{\mathrm{DL}} = \sqrt{\frac{2N-5}{(N-1)(N-3)}}.$$

Here RMS denotes a root second moment, not exactly a standard deviation: the DL mean is $-1/(N-1)$, although this distinction is negligible for the 32-bit experiments. For large N, the linear and DL RMS values are approximately $N^{-1/2}$ and $\sqrt{2/N}$, respectively. The

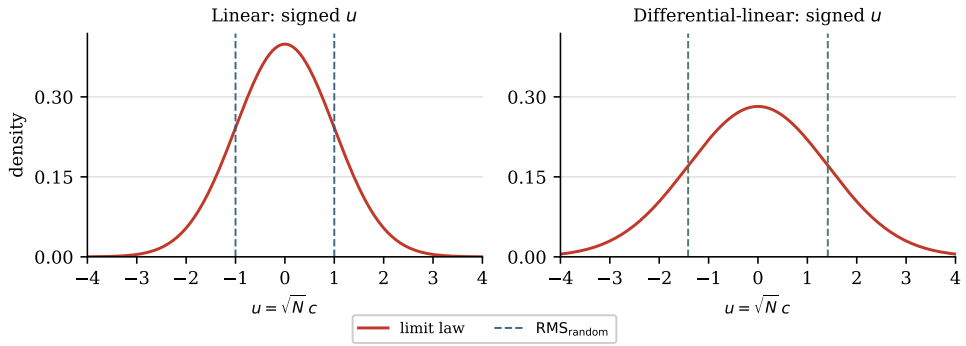


Figure 5: Random-permutation correlation laws in the rescaled variable $u = \sqrt{N}\,c$. Dashed lines mark the root-second-moment references.

factor $\sqrt{2}$ in the DL scale comes from averaging over $N/2$ unordered pairs, with each pair counted twice in the full-codebook sum. An experiment with s independent unordered derivative pairs has an empirical DL variance of about $1/s$. The full-codebook experiment has $s = N/2$ distinct pairs and therefore matches the leading term $2/N$. Statements of the coarser scale $2^{-n/2}$ either suppress the constant factor $\sqrt{2}$ or use a model with N independent pair observations; the latter cannot arise from one fixed non-zero difference under one permutation. In the same large-N normal approximation, the mean absolute DL correlation is

$$\mathbb{E}[|D|] \approx \sqrt{\frac{2}{\pi}}\,\mathrm{RMS}^{\mathrm{DL}}_{\mathrm{random}} \approx \frac{2}{\sqrt{\pi N}}.$$

Figure 5 shows these laws in the rescaled variable $u = \sqrt{N}\,c$. Rescaling by $\sqrt{N}$ removes the block size, so one pair of curves stands for every n. In these units the linear law tends to a standard normal and the DL law to a normal of standard deviation $\sqrt{2}$, so the DL law is wider by exactly that factor. The exact discrete masses of Equation 4 and Equation 5 converge to these limits and are already visually indistinguishable from them at $n = 32$, and the DL mean $-1/(N-1)$ vanishes as $-1/\sqrt{N}$ in this variable.

The figure gives the laws as the theorems state them. A verified RMS above the reference suggests structure beyond the ideal random-permutation model; quantifying statistical significance additionally requires the uncertainty from the finite verification sample.

6.3 Candidate Discovery and Confirmation

Our full-codebook analysis had two phases.

Phase 1: Discovery sweep. Write $N = 2^n$. A linear distinguisher that survives many rounds usually has a mask of low Hamming weight on at least one side. A low-weight mask constrains few state bits, so the round function needs more rounds to spread it across the state. The input difference of a DL distinguisher behaves the same way. We used this observation as a search heuristic, not as a statement about which distinguishers are best.

In the low-weight sweeps, we therefore fixed one side to the non-zero masks or differences of Hamming weight at most w, for a small bound $w \le n$, and left the other side free. The fixed side then offers

$$q = \sum_{j=1}^{w} \binom{n}{j}$$

candidates, so w controls the cost of the sweep directly. Most experiments in this paper used $w = 2$, which gives $q = n + \binom{n}{2}$, that is $q = 528$ for $n = 32$. Even with this simple strategy, the sweeps improved published trail-based results for several small-block ciphers and supplied exact distinguishers where none had been reported. The FWHT evaluated all masks on the free side simultaneously; after excluding the trivial zero mask, the restriction therefore applied only to the fixed side. For DL analysis, we excluded $\beta = 0$ because $\mathrm{Cr}_k^{\mathrm{DL}}(\delta, 0) = 1$ for every non-zero input difference δ. For linear analysis, a zero free mask gives correlation zero when the fixed-side mask is non-zero. Each sweep orientation therefore evaluated $q(N-1)$ nontrivial fixed-side/free-side pairs. Targeted sweeps instead used an explicitly specified fixed-side set, with q denoting its cardinality and the free side still evaluated exhaustively.

The low-weight heuristic gives no guarantee: a distinguisher whose two sides both have high weight cannot be reached with a small w, and we do not claim that our sweeps return the strongest distinguisher a target admits. Section 7 gives these results per target.

For every fixed-side candidate and each of K_{sweep} sampled keys, we evaluated the full-codebook correlation spectrum over the free side. This placed every candidate pair on a common sample of keys for ranking.

Let $k_1, \ldots, k_{K_{\mathrm{sweep}}}$ denote the sampled sweep keys, where $K_{\mathrm{sweep}} \geq 2$. Let $\pi = (\alpha, \beta)$ denote a linear pair or $\pi = (\delta, \beta)$ a DL pair. For $1 \leq i \leq K_{\mathrm{sweep}}$, let $c_i(\pi)$ denote the corresponding correlation under key k_i. We defined the per-key magnitude $a_i(\pi) = |c_i(\pi)|$ and its sample mean as

$$\overline{a}(\pi) = \frac{1}{K_{\mathrm{sweep}}} \sum_{i=1}^{K_{\mathrm{sweep}}} a_i(\pi).$$

The discovery phase supports three ranking modes, each suited to a different notion of a useful key-dependent correlation.

RMS mode. RMS mode ranks candidates by

$$R(\pi) = \left(\frac{1}{K_{\mathrm{sweep}}} \sum_{i=1}^{K_{\mathrm{sweep}}} c_i(\pi)^2 \right)^{1/2}.$$

It measures the aggregate correlation energy over the sampled keys and therefore does not allow opposite signs to cancel. It is appropriate when the objective is a large key-averaged second moment, irrespective of how that energy is distributed among keys.

Stable mode. Stable mode uses the sample standard deviation of the per-key magnitudes,

$$s_a(\pi) = \left(\frac{1}{K_{\mathrm{sweep}} - 1} \sum_{i=1}^{K_{\mathrm{sweep}}} \left(a_i(\pi) - \overline{a}(\pi)\right)^2 \right)^{1/2}.$$

It ranks candidates by

$$S(\pi) = \overline{a}(\pi) - s_a(\pi).$$

This favors a large typical magnitude while penalizing variation, so it discounts candidates whose apparent strength sits in a small set of keys.

Lower-tail mode. Fix a tail fraction $\tau \in (0, 1]$, put $m = \lceil \tau K_{\mathrm{sweep}} \rceil$, and order the magnitudes as $a_{(1)}(\pi) \leq \cdots \leq a_{(K_{\mathrm{sweep}})}(\pi)$. Lower-tail mode ranks candidates by

$$T_\tau(\pi) = \frac{1}{m} \sum_{i=1}^{m} a_{(i)}(\pi).$$

It directly measures the average strength in the weakest fraction of the sampled keys. Unlike stable mode, it does not penalize exceptionally strong keys; instead, weak-key behavior alone lowers the score.

For each mode, we retained candidates with large scores for confirmation. Restricting the fixed side kept the discovery search tractable.

Choice of mode. Most distinguishers reported in this paper were discovered in stable mode, and we name the mode wherever another one was used. The choice matters because here the ranking criterion is a parameter rather than a property of the search. Automated searches built on mixed-integer programming or satisfiability rank candidates by the weight of a dominant trail, so they order candidates by an estimate of one contribution. A DL correlation is a signed sum over many trails, and cancellation among them changes both the size and the sign of the result, so a ranking built on one trail can order candidates wrongly. Ranking by a quantity measured over sampled keys avoids that estimate, and it lets the criterion match what an attacker needs, whether that is average strength, stability across keys, or strength on the weakest keys. The cost is that every candidate must be evaluated exactly, which confines this search to block sizes where a full codebook is affordable.

Relation to black-box property detection. Like our discovery phase, the surrogate-differentiation algorithms proposed by Dinur et al. [DDK+23] detect differential and linear properties without a trail model. However, the methods apply to different ranges of block sizes and biases. Their linear algorithm finds every approximation of bias at least p with time, data, and memory $O(2^{n/2}p^{-2})$, provided that $p \geq 2^{-n/4}$. This condition ensures that the output of a derived function can be reduced to $n/2 + \log(p^{-2})$ bits without exceeding the block size. For $n = 32$, the condition requires a bias of at least 2^{-8}. In contrast, correlations near 2^{-13}, as reported in our experiments, correspond to biases near 2^{-14} and would require a 44-bit output. These approximations therefore lie outside the algorithm's stated range, whereas our method evaluates them using the full 32-bit codebook, which occupies 16 GiB when stored as 32-bit words. For larger blocks and stronger biases, surrogate differentiation can be substantially less expensive than full-codebook evaluation. For example, with a 64-bit block and bias 2^{-8}, its complexity bound is $O(2^{48})$, compared with the 2^{64} inputs required for a full codebook. The cited work leaves the extension to DL properties as an open problem, so its algorithms do not provide a corresponding comparison for our DL results.

Phase 2: Fixed-pair confirmation. After Phase 1 selected a high-scoring pair, we fixed both sides of the correlation. We wrote the selected pair as $\pi^* = (\alpha^*, \beta^*)$ for linear analysis or $\pi^* = (\delta^*, \beta^*)$ for DL analysis. For linear correlation, we evaluated the scalar

$$\mathrm{Cr}_k^{\mathrm{Lin}}(\alpha^*, \beta^*) = \frac{1}{2^n} \sum_{x \in \mathbb{F}_2^n} (-1)^{\alpha^* \cdot x + \beta^* \cdot E_k(x)}.$$

For DL correlation, we evaluated $\mathrm{Cr}_k^{\mathrm{DL}}(\delta^*, \beta^*)$ from Equation 3 using the same full-codebook definition. We used the symmetry between x and $x + \delta^*$ to avoid redundant DL evaluations. This optimization did not change the correlation value.

Let K_{conf} denote the number of confirmation keys sampled independently of the sweep keys. This scalar evaluation made $K_{\mathrm{conf}} \gg K_{\mathrm{sweep}}$ practical on a workstation. Let d_i denote the selected correlation under the i-th confirmation key. We reported the confirmation RMS value

$$\mathrm{RMS}_{\mathrm{conf}}(\pi^*) = \left(\frac{1}{K_{\mathrm{conf}}} \sum_{i=1}^{K_{\mathrm{conf}}} d_i^2 \right)^{1/2}.$$

The mean square avoids cancellation between correlations with opposite signs.

That cancellation is itself informative, so we record it separately. Writing $\bar{d}$ for the mean of the signed confirmation values, the mean–variance identity implies $\mathrm{RMS}^2 = \bar{d}^2 + \mathrm{Var}(d)$, so

$$\rho = \frac{\bar{d}^2}{\mathrm{RMS}^2} \in [0, 1]$$

is the share of the correlation's power that survives averaging over the sampled schedule inputs when the RMS is nonzero. The schedule inputs are master keys for the ordinary ciphers and (K, T_1, T_2) tuples for the variable-tweak BEANIE runs. It equals one exactly when the sampled correlation is constant, and it equals zero exactly when the signed mean is zero, as it is for an exactly balanced sign distribution of equal-magnitude correlations. We report ρ as a diagnostic only, never as a ranking score. A magnitude-based distinguishing test compares $|\hat{d}|$ against a threshold without using its sign, so a small ρ does not by itself preclude distinguishing. The sign distribution can also provide distinguishing information. Subsection 8.8 collects what ρ shows across the five targets.

Selection bias. Phase 1 ranked $q(N-1)$ empirical discovery scores from the same K_{sweep} keys and retained pairs with large observed scores. Even when candidate pairs have equal population scores under the chosen ranking mode, sampling variation preferentially raises the retained scores. This effect is called selection bias or the *winner's curse*.

Phase 2 evaluated each selected pair on new, independent keys. Conditional on the Phase 1 result, the mean square

$$\frac{1}{K_{\mathrm{conf}}} \sum_{i=1}^{K_{\mathrm{conf}}} d_i^2$$

is an unbiased estimator of the selected pair's key-averaged second moment. Its square root is the reported confirmation RMS, but the square root is not generally an unbiased estimator of the population RMS. The confirmation RMS has finite-sample variation, but it does not reuse the keys that selected a large score. With RMS discovery, the sweep and confirmation values target the same key-averaged second moment but use independent samples; with stable or lower-tail discovery, they intentionally summarize different key-dependent properties. In all cases, the independently estimated confirmation RMS need not coincide with the discovery score. The random-permutation laws from Theorem 2 and Theorem 3 provide the comparison scale for interpreting the confirmation RMS.

Linear correlation analysis (Phase 1 detail). Fix an input mask $\alpha \in \mathbb{F}_2^n$ and a key k. Iterate over all plaintexts $x \in \mathbb{F}_2^n$ and accumulate the sign-encoded plaintext parity into a ciphertext-indexed array:

$$\mathcal{L}[E_k(x)] = (-1)^{\alpha \cdot x}.$$

One call to `fwht_i32` on this list $\mathcal{L}$, followed by normalization by 2^{-n}, yields $\mathrm{Cr}_k^{\mathrm{Lin}}(\alpha, \beta)$ for all output masks β simultaneously. If repeated queries use the same fixed key, the ciphertext look-up table E_k can be cached once and reused for several input masks. This cache avoids repeating the full plaintext sweep.

DL correlation analysis (Phase 1 detail). Fix an input difference $\delta \in \mathbb{F}_2^n$ and a key k. Sweep all 2^n plaintexts x and accumulate the output-difference histogram

$$N_{k,\delta}[\gamma] = |\{E_k(x) + E_k(x + \delta) = \gamma\}|.$$

One call to `fwht_i32` on this length-2^n integer histogram, followed by normalization by 2^{-n}, computes $\mathrm{Cr}_k^{\mathrm{DL}}(\delta, \beta)$ for all output masks β simultaneously, as specified in Equation 3. This convention matches the signed DLCT in [HDE24]. The object to transform is the

histogram, not the raw list of output differences; that choice is exactly what Equation 3 requires and is what makes one WHT produce all free masks at once.

7 Applications to Small Block Ciphers

This section instantiates the full-codebook method for five cipher targets: SIMON-32, KATAN-32, BEANIE, KeeLoq, and RC5-16. Each case study reports confirmed correlation and RMS magnitudes for reduced-round distinguishers, measured over independently sampled schedule inputs. Appendix E collects the cipher specifications and the state and key encodings used below. The per-sample distribution behind every reported value is collected in Appendix C.

7.1 Application to SIMON-32

We applied the method to reduced-round SIMON32/64, using the first rounds and the standard 64-bit-key schedule specified in Subsection E.1. Prior cipher-specific DL studies of SIMON32/64 include Chen and Zhang [CZ18] and Zhang et al. [ZLZ23]. Zhou, Wang, and Hu [ZWH24] and Chen et al. [CXZQ25] developed automated searches for DL distinguishers of SIMON-like ciphers. Throughout this case study, we report nonzero correlation and RMS magnitudes as 2^{-e}, with e rounded to two decimal places.

As a consistency check, our exact linear-support tool also audited the 16-round SIMON-64 entry of Chen et al. [CXZQ25]. The published tuple has input difference `(0x20,0x88)` and output mask `(0x1000000,0x4400004)`. Its final 5-round linear suffix has correlation zero, consistent with our GPU result at the noise floor. Inserting one missing zero in each output word, to `(0x10000000,0x44000004)`, yields a valid trail with the claimed squared correlation 2^{-10}. This correction suggests a transcription error rather than a defect in the search model.

Table 5: Published and discovered distinguishers for round-reduced SIMON-32. Shading marks improvements over published or independently remeasured correlations.

		Published distinguisher				Ours (confirmed RMS)			
Type	#R	Input	Output	Correlation	Source	Input	Output	RMS	Figure
DL	10	–	–	–	–	0x00002000	0x00101004	$2^{-4.96}$	Figure 10a
	11	0x00080022	0x00400010	$2^{-7.91}$ ($2^{-7.91}$†)	[CXZQ25]	0x00002000	0x00040013	$\mathbf{2^{-6.96}}$	Figure 10b
	12	0x00080020	0x08000200	$2^{-10.84}$ ($2^{-10.84}$†)	[ZLZ23]	0x40000001	0x00080022	$\mathbf{2^{-8.95}}$	Figure 10c
	13	0x01000440	0x08002200	$2^{-11.99}$ ($2^{-10.94}$†)	[CXZQ25]	0x01000441	0x08002600	$\mathbf{2^{-10.94}}$	Figure 10d
	14	0x01000645	0x80002002	$2^{-14.48}$ ($2^{-14.36}$†)	[CXZQ25]	0x80000002	0x0001c144	$\mathbf{2^{-13.91}}$	Figure 10e
	15	0x00080022	0x00040111	$2^{-15.40}$ ($2^{-15.20}$†)	[CXZQ25]	0x00080022	0x00040111	$2^{-15.20}$	Figure 10f
Lin	10	0x01014404	0x10004404	$2^{-11.60}$ ($2^{-11.91}$†)	[AAA+15]	0x21414404	0x10004404	$\mathbf{2^{-11.60}}$	Figure 11b
	11	–	–	$2^{-15.00}$	[KLT15]	0x08020000	0x00080006	$\mathbf{2^{-13.29}}$	Figure 11c
	12	–	–	–	–	0x10000000	0x10000400	$2^{-13.61}$	Figure 11d
	13	0x00200000	0x00002000	$2^{-14.50}$ ($2^{-14.09}$†)	[SHS+17]	0x00200000	0x00002000	$2^{-14.09}$	Figure 11e
	14	0x00800000	0x80002000	$2^{-15.29}$	[AAA+15]	0x08000000	0x00080002	$\mathbf{2^{-14.92}}$	Figure 11f

Notes. Published correlations are experimental RMS values where available, otherwise reported RMS-equivalents. Input is a mask for linear analysis and a difference for DL. Our RMS values use 10,000 independent keys; † marks our full-codebook remeasurement of the cited distinguisher. The 15-round DL distinguisher was rediscovered, not newly identified.

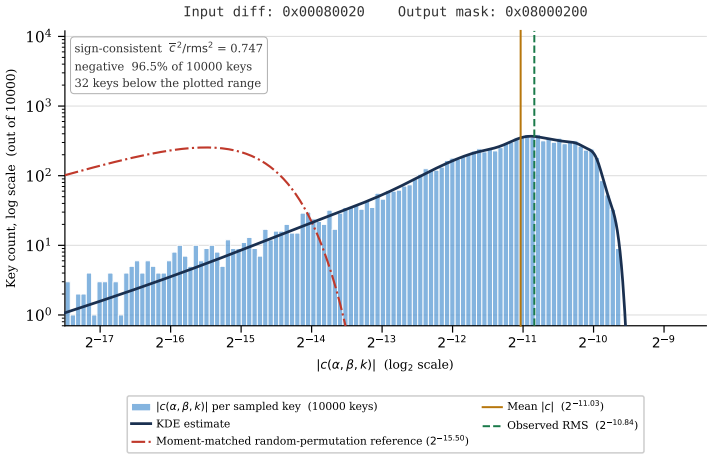
For the 13-round published pair, Chen et al. estimated $2^{-11.99}$ and reported $2^{-10.95}$ experimentally over 100 keys, while our full-codebook confirmation over 10,000 independent keys gave RMS $2^{-10.94}$ [CXZQ25].

At 15 rounds, a targeted RMS sweep fixed the published input difference `0x00080022` and evaluated all $2^{32} - 1$ nonzero output masks over 1,000 keys. The highest-scoring mask was `0x00040111`, exactly the mask reported by Chen et al. [CXZQ25], with a discovery RMS of $2^{-15.12}$. Earlier sweeps with 50 and 200 keys selected different masks whose independently confirmed RMS values were near the random-permutation reference.

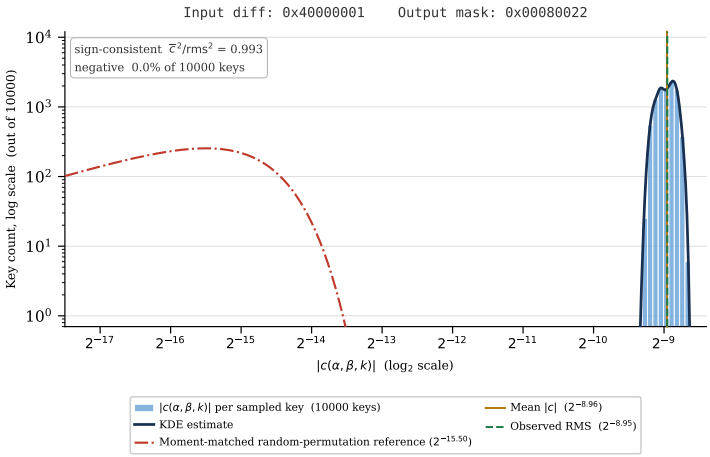
Recovering the published mask with a larger discovery sample is consistent with the selection bias discussed in Subsection 6.3: exhaustive coverage of output masks does not remove uncertainty from sampling keys.

In full-codebook confirmation over 10,000 independent keys, the recovered pair's correlation was positive on 72.71% of the keys, with a 95% Wilson interval of [71.83%, 73.57%], compared with approximately half under Theorem 3. Thus, even a test that declares the cipher when the full-codebook correlation is positive has empirical support as a weak distinguisher. This observation does not establish a limited-data attack.

Figure 6 places our 12-round DL distinguisher beside the published one at the same round count, over the same 10,000 keys. The published correlation ranges over 13.8 bits across keys, and its weakest 5% of keys fall 2.91 bits below its RMS. Ours ranges over 0.59 bits, with its weakest 5% only 0.21 bits below. By the argument of Subsection 8.9, the data a key recovery needs is therefore predictable from our reported RMS on almost every key, and is not predictable from the published one. This is what the stable score of Subsection 6.3 selects for, because it subtracts the spread across keys instead of rewarding a large average alone.



(a) Published [ZLZ23]



(b) Ours

Figure 6: The two 12-round DL pairs of Table 5, each evaluated on the complete codebook for the same 10,000 keys and drawn on identical axes.

For the 10-round linear row, a fixed-output sweep at the published output mask found input mask `0x21414404`. Over 10,000 independent keys, its RMS was $2^{-11.60}$, compared with $2^{-11.91}$ for the published pair under the same procedure. Figure 10 and Figure 11 show the per-key distributions. The masks differ between panels, so these are not controlled round-by-round decay experiments.

7.2 Application to KATAN-32

The register-based KATAN-32 provides a contrasting target to the Feistel structure of SIMON-32 [CDK09]. All local measurements begin at round zero and use the standard 80-bit-key schedule and state encoding in Subsection E.2. Shi et al. reported both an 84-round single characteristic with correlation 2^{-15} and a stronger 98,264-trail hull on the same external masks with potential $2^{-27.93}$ [SHSS16]. Table 6 compares the hull's RMS-equivalent value with our confirmation RMS; the single characteristic is not listed separately. To the best of our knowledge, the 80- and 84-round rows are the first reported exact DL distinguishers for KATAN-32: they supply explicit input differences and output masks, and their full-codebook correlations are independently confirmed over fresh keys.

Table 6: Published bounds and distinguishers for round-reduced KATAN-32, compared with our exact measurements.

		Published result or bound				Ours (confirmed RMS)			
Type	#R	Input	Output	Correlation	Source	Input	Output	RMS	Figure
DL	80	–	–	–	–	0x00008010	0x02024000	$\mathbf{2^{-12.16}}$	Figure 12a
	84	–	–	–	–	0x00008010	0x20044000	$\mathbf{2^{-12.18}}$	Figure 12b
	120	–	–	$\lvert c\rvert \le 2^{-21}$	[CDK09]	–	–	–	–
Lin	42	–	–	$\lvert c\rvert \le 2^{-5}$	[CDK09]	–	–	–	–
	84	0x84400010	0x20040000	$2^{-13.97}{}_{*}$ ($2^{-14.74}{}_{\dagger}$)	[SHSS16]	0x8d400010	0x20040000	$2^{-14.79}$	Figure 12d

Notes. Published quantities are expressed as absolute correlations. The 120-round DL bound is converted from the designers' half-correlation convention and does not specify an exact distinguisher. Here $*$ denotes the RMS-equivalent $\sqrt{P}$ of the published hull potential $P = 2^{-27.93}$, and † our independent full-codebook RMS remeasurement of the cited distinguisher. Our confirmations use 100 independent keys for 80-round DL and 1,000 otherwise.

Figure 12 shows the selected pairs alongside an independent confirmation of the published 84-round linear pair.

7.3 Application to BEANIE

We studied BEANIE in the designers' U-shape attacker model [GHM+25]. For a master key K and distinct tweaks T_1, T_2, let E_{K,T_1,R_1} encrypt the final R_1 rounds of the five-round parent, including whitening, and let D_{K,T_2,R_2} invert its final R_2 rounds under the second tweak. The observable map is $U(P) = D_{K,T_2,R_2}(E_{K,T_1,R_1}(P))$; the intermediate ciphertext is hidden. Both tweakey schedules use five rounds and the same master key, with the derivation specified in Subsection E.3. The designers claim practical security against 2^{80} time and 2^{40} data for a 5+5-round U-shape, and recommend 7+7 rounds for 128-bit security. They explicitly do not claim security outside this U-shape setting [GHM+25]. Their best 3+2 linear-hull estimate has expected linear potential (ELP) $2^{-21.9}$.

Table 7 compares the correlation-compatible designer results with our exact full-codebook measurements. The local rows use independently sampled (K, T_1, T_2) tuples with distinct tweaks. Unlike the published ELP estimate, each local RMS is calculated directly from all 2^{32} inputs per tuple. Thus, the published and local 3+2 rows describe the same U-shape linear-hull category, but not the same mask pair or estimation method. The selected 2+3 and 3+2 linear distinguishers have nearly equal RMS, as expected from U-shape mirroring, despite the asymmetry of the designers' finite clustering estimates. No conventional DL cryptanalysis appears in the design paper. To the best of our knowledge,

the 2+3 and 3+2 DL rows are therefore the first reported exact DL distinguishers for BEANIE. Figure 13 shows the per-tuple distribution of each selected pair.

Table 7: Published and measured U-shape correlations for BEANIE.

		Published result or bound				Ours (verified)			
Type	#R	Input	Output	Correlation	Source	Input	Output	RMS	Figure
DL	2+3	–	–	–	–	0x00800400	0xe070000c	$2^{-4.95}$	Figure 13a
	3+2	–	–	–	–	0x00000080	0x20000700	$2^{-8.46}$§	Figure 13b
Lin	3+2	–	–	$2^{-10.95}$∗	[GHM+25]	0xc0e00d00	0x00000001	$\mathbf{2^{-10.75}}$	Figure 13d
	2+3	–	–	–	–	0x00004020	0x000440b0	$\mathbf{2^{-10.68}}$	Figure 13c

Notes. R_1+R_2 counts the forward and inverse suffix rounds of the five-round parent, including final whitening. The value marked ∗ is $\sqrt{\text{ELP}}$ for the designers' estimated ELP $2^{-21.9}$; no input or output masks are specified. Our full-codebook RMS confirmations use 1,000 independent (K, T_1, T_2) tuples with $T_1 \neq T_2$. The marker § indicates a signed correlation varying around zero, also when only K varies at fixed tweaks; see Figure 9.

These results characterize reduced U-shape instances, not the 5+5 practical-security target.

7.4 Application to KeeLoq

For KeeLoq, we studied reduced-cycle linear and DL correlations under the standard 64-bit key schedule and cycle convention in Subsection E.4. Its 64-cycle key repetition and five-input nonlinear feedback function (NLF) also connect the exact measurements to the linear slide attack of Bogdanov [Bog07].

Table 8 records the only explicit prior linear relation whose masks and correlation can be directly verified from the original cryptanalysis. It is a 32-cycle internal-state approximation from $Y^{(32)}$ to $Y^{(64)}$, embedded in the source's 64-cycle slide-pair construction, rather than a conventional 528-cycle plaintext/ciphertext approximation. Equation (6) uses the $Y^{(32)}$ mask 0x00000203 (bits 0, 1, and 9) and the $Y^{(64)}$ mask 0x00000001, together with the source's c_0 preprocessing term and key offset $k_{16} \oplus k_{32}$; its correlation magnitude is 2^{-2} [Bog07]. We nominated fixed mask pairs with 20-key GPU sweeps and confirmed each on the complete 2^{32}-input codebook for 1,000 fresh, independently sampled 64-bit keys. Figure 14 and Figure 15 show the per-key distributions. The checked literature likewise provides no classical DL input-difference/output-mask/correlation tuple. To the best of our knowledge, the 64-, 100- and 128-cycle rows are therefore the first reported exact DL distinguishers for KeeLoq. At 32 cycles the 20-key RMS discovery sweep returns 23 mask pairs tied at the same RMS maximum, so no single pair among them is distinguished by the search. Table 8 lists one representative of that set, chosen because it is the pair whose confirmed RMS exceeds the published correlation. We confirmed two of the 23 pairs; both have $|c| = 2^{-1.00}$ for every sampled key, with only the sign depending on the key. Figure 16 shows these two representatives, one from each sweep orientation.

The exact 32-cycle analysis also led us to revisit the correlation phase of Bogdanov's basic linear slide attack [Bog07]. Number the states of its 64-cycle permutation by $Y^{(t)} = (y_{31}^{(t)}, \ldots, y_0^{(t)})$, where $Y^{(64)}$ is the output available to the attacker, and let $j \in \{0, \ldots, 15\}$ denote the step of the key peel. The right shift gives

$$y_{31}^{(32+j)} = y_{j-1}^{(64)} \quad (j \geq 1), \qquad y_{26}^{(32+j)} = y_{j-6}^{(64)} \quad (j \geq 6), \qquad y_{20}^{(32+j)} = y_{j-12}^{(64)} \quad (j \geq 12).$$

Thus, these NLF inputs become known output bits as the peel advances. An exhaustive evaluation of the 32-entry NLF truth table gives four regimes. The published 2^{-2}-correlation approximation remains necessary at step 0, while conditioning on $y_{31}^{(32+j)} = 0$ gives correlation 2^{-1} at steps 1–5. At steps 6–11, the NLF is determined exactly for 8 of the 16 configurations of the known taps; at steps 12–15, it is determined exactly on every

pair. Courtois, Bard, and Bogdanov had already described the general use of forward and backward state exposure and affine reduction of an NLF with one unknown input [CBB08]. Our refinement uses the staggered ciphertext-side taps inside this particular key peel.

We validated complete 64-bit recovery on genuine pseudo-slide groups, that is, groups arising from the correct partial-key and round-output guess. Over 50,000 independently sampled master keys and initial inputs, the refined peel recovered 49,068 keys from groups of 128 pairs, for a success rate of 98.136% and a 95% Wilson interval of [98.014%, 98.251%]. Our implementation of the published estimator recovered 46,807 of the same 50,000 keys from groups of 256 pairs, for a success rate of 93.614%. Under Bogdanov's operation model, the work per guessed group falls from 12.28 to approximately 3.53 KeeLoq encryptions, which changes the modeled complexity of the basic attack from $2^{50.6}$ to $2^{48.82}$, a factor of 3.48. The data and memory requirements remain 2^{32}, and this refinement does not reduce the separate 2^{37}-encryption advanced attack because its cycle-structure phase dominates.

Table 8: Published and independently confirmed distinguishers for KeeLoq.

		Published result or bound				Ours (RMS)			
Type	#R	Input	Output	Correlation	Source	Input	Output	RMS	Figure
DL	64	–	–	–	–	0x00000080	0x00000003	$2^{-1.46}$	Figure 14a
	100	–	–	–	–	0x00080008	0x01001086	$2^{-7.32}$	Figure 14b
	128	–	–	–	–	0x00000001	0x0200210c	$2^{-14.07}$	Figure 14c
Lin	32	0x00010203*	0x00000001	2^{-2} ($2^{-2.00}$ † §)	[Bog07]	0x00800000	0x00800881	$\mathbf{2^{-1.00}}$§	Figure 16a
	64	–	–	–	–	0xc0000000	0x20c060c4	$2^{-4.00}$	Figure 15a
	100	–	–	–	–	0x18000000	0x04008668	$2^{-10.00}$	Figure 15b

Notes. The published linear relation spans $Y^{(32)}$ to $Y^{(64)}$ within the source's 64-cycle slide construction. The mask marked $*$ restores bit 16, omitted by the substitution $y_{16}^{(32)} = c_0 \oplus k_{16}$; without it the correlation is zero. The key-dependent offset $k_{16} \oplus k_{32}$ remains outside the mask columns. Our full-codebook confirmations use 1,000 independent master keys; † marks the remeasurement of the published relation. The marker § means that the magnitude is constant across sampled keys and only the sign depends on the key.

The reduced-cycle correlations in Table 8 do not themselves constitute a full-round attack on the 528-cycle cipher.

7.5 Application to RC5-16

We studied RC5-16/r/16, with 32-bit blocks and 128-bit master keys, using initial whitening followed by r complete rounds and the key schedule expanded for that same r; see Subsection E.5. Its rotation amounts depend on intermediate data, and hence on both the plaintext and the key [Riv94]. Unlike a fixed rotation, each such operation requires a MILP/SAT trail model to account for the rotation selector as well as the rotated value. The black-box search avoids this modelling step and yields, to the best of our knowledge, the first exact DL distinguishers for RC5-16.

Table 9 reports our linear and DL distinguishers through six rounds. Linear discovery used RMS over 20, 50, and 20 keys at four, five, and six rounds, respectively. The first two sweeps fixed one-bit input masks; the six-round sweep fixed output masks of Hamming weight at most two. DL discovery fixed one-bit input differences and used 30, 20, and 20 keys, with the stable score at five rounds and RMS otherwise. Figure 17 shows the distributions behind these summaries.

The earlier linear estimates and measurements require a distinction in round boundaries. Selçuk evaluates $R_0[0] \oplus L_{2r}[0]$ in a half-round notation, with the Kaliski–Yin estimate $|c| = w^{-(r-1)}$ [KY95, Sel00]. Writing $d = r-1$, this is the upper-word relation $B_0[0] \oplus B_d[0]$ under Subsection E.5, with input and output masks different from ours. The published values are therefore contextual benchmarks, not matched-mask reproductions.

Table 9: RC5-16 distinguishers and published linear benchmarks.

		Published result or estimate				Ours (confirmed RMS)			
Type	#R	Input	Output	Correlation	Source	Input	Output	RMS	Figure
DL	4	–	–	–	–	0x00000001	0x00000001	$2^{-7.51}$	Figure 17a
	5	–	–	–	–	0x00000001	0x00000004	$2^{-10.84}$	Figure 17c
	6	–	–	–	–	0x00000001	0x00000001	$2^{-13.78}$	Figure 17e
Lin	4	0x00010000	0x00010000	2^{-16} ($2^{-11.1}$∗)	[KY95, Sel00]	0x00000001	0x00000001	$2^{-11.21}$	Figure 17b
	5	0x00010000	0x00010000	2^{-20} ($2^{-13.6}$∗)	[KY95, Sel00]	0x00000001	0x00001800	$2^{-13.40}$	Figure 17d
	6	0x00010000	0x00010000	2^{-24} ($2^{-15.3}$∗)	[KY95, Sel00]	0x00020000	0x00000001	$\mathbf{2^{-14.22}}$	Figure 17f

Notes. Input is a difference for DL and a mask for linear analysis. Published values give the Kaliski–Yin estimate and, in parentheses (∗), Selçuk's mean absolute correlation converted from bias. The source uses $r = \#R + 1$ and the intermediate boundary described above. Both studies exhaust 2^{32} plaintexts per key: Selçuk used 500 random keys; our independent confirmations use 1,000 for six-round linear analysis and 10,000 otherwise. Larger key samples refine across-key statistics, not fixed-key accuracy. Our linear mean magnitudes are $2^{-11.62}$, $2^{-13.81}$, and $2^{-14.64}$ at four, five, and six rounds, respectively. Only the six-round mean exceeds the published benchmark; the masks and parent key schedules differ.

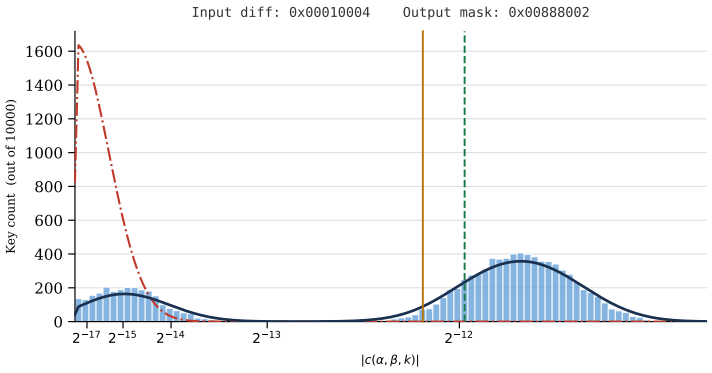
8 Key Dependence and the Limits of Summary Statistics

Section 7 summarizes each distinguisher by the confirmation RMS of Subsection 6.3. That number summarizes a whole distribution, namely one correlation for each sampled key. A summary is faithful only when the distribution behind it has a simple shape. We therefore inspected the measured distributions rather than reporting the summary alone. Most of them concentrate around a single value, and the RMS describes them well. The following example shows why this need not hold.

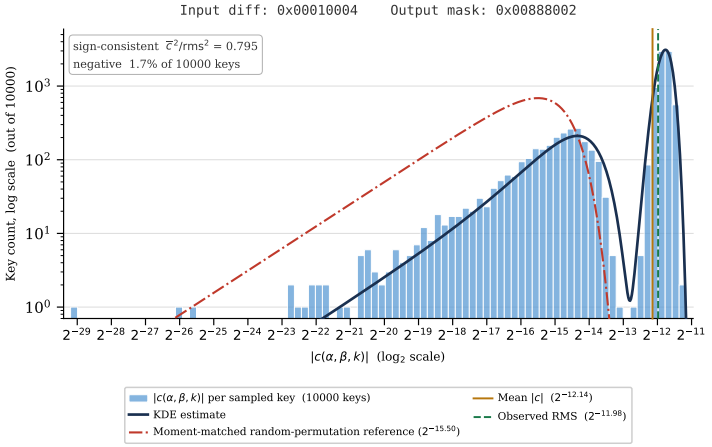
Figure 7 shows the 13-round DL distinguisher of SIMON-32 with input difference 0x00010004 and output mask 0x00888002, measured over 10,000 independently sampled master keys. Its distribution has two peaks. About 3 keys in 4 give a correlation near $2^{-11.8}$. The remaining keys give a correlation near $2^{-14.6}$, which is about 7 times smaller. The measured RMS of $2^{-11.98}$ lies inside the strong peak and carries no information about the weak one. This pair is not a competitive 13-round distinguisher, and Table 5 does not list it. We study it here because its key dependence is unusually clear and because the explanation transfers to other targets.

The two peaks are not an artifact of one ranking criterion. With the input difference restricted to Hamming weight at most 2, the leading 13-round DL distinguisher from each of the three modes of Subsection 6.3 shows the same shape. All three reach RMS $2^{-11.98}$, and in all three 23% of the 10,000 keys lie more than 2 bits below it. They differ only in the extreme left tail, the side that matters for an attacker, where lower-tail ranking returns the pair whose worst key is the least extreme, at $2^{-23.54}$ against $2^{-29.00}$ for stable ranking. Widening the search to input differences of Hamming weight 3 returns stronger 13-round DL distinguishers, which Table 5 reports. The distribution figures share one visual key. A dash-dot red curve shows the moment-matched half-normal reference of the ideal random-permutation model, and a dark blue curve shows the Gaussian kernel-density estimate. A green dashed line marks the observed RMS, and an amber solid line the mean $|c|$. An inset reports the sign statistic ρ of Subsection 6.3 together with the share of keys whose signed correlation is negative. A figure that shows the same measurements on both a linear and a logarithmic axis carries this key on one of its two panels only.

This section proceeds in four steps. Subsection 8.1 states what key dependence means for a correlation, and Subsection 8.2 presents a procedure that finds the key bits responsible for it. Four worked cases follow. Subsection 8.3 identifies the 2 round-key bits behind the two peaks of Figure 7, and Subsection 8.4 applies the same steps to a published distinguisher whose key dependence turns out to be real but small. Subsection 8.5 shows why cancellation in a narrow weight window must not be read as a zero coefficient, and Subsection 8.6



(a) Linear axes



(b) Logarithmic axes

Figure 7: Two views of the same 10,000 measurements. The linear axes separate the two clusters and show their relative sizes. The logarithmic axes make the sparse tail readable, where 9 keys fall below 2^{-21} and reach $2^{-29.00}$, and place the random-permutation reference of Subsection 6.2 on the same axis.

presents a case whose leading key-dependent order does cancel. Subsection 8.7 states what the four cases together show about the procedure. Subsection 8.8 collects the key-dependent behaviour of all five targets, and Subsection 8.9 states what these observations mean for reporting a distinguisher.

8.1 Correlation as a Function of the Round Keys

Consider a cipher that runs r rounds and adds a round key k_i to the state in round i. Write $K = (k_0, \ldots, k_{r-1})$ for the vector of all round keys, and write $k_i[j]$ for bit j of k_i. The key schedule derives K from the master key. Fix an input difference δ and an output mask β. The correlation $\mathrm{Cr}^{\mathrm{DL}}(\delta, \beta)$ of Equation 3 is then a real-valued function of K alone, which we write as $\mathrm{Cr}(K)$, leaving the fixed pair implicit.

Every real-valued function of a binary vector has a unique Walsh expansion. For Cr that expansion reads

$$\mathrm{Cr}(K) = \sum_{\mu} c_\mu \, (-1)^{\mu \cdot K}, \tag{6}$$

where μ runs over all binary vectors of the same length as K, and each c_μ is a real number.

Definition 3 (Key mask and key-mask coefficient)**.** In Equation 6, a vector μ is a *key mask*, and the associated real number c_μ is its *key-mask coefficient*.

Equation 6 is a rewriting of Cr, not an assumption about it. Its value comes from making the contributions of individual key parities explicit. The coefficient c_0 of the zero key mask is the average of Cr over all independent round-key vectors, because every other term changes sign over that space. This expansion characterizes round-key dependence: the correlation is the same for every round-key vector exactly when $c_\mu = 0$ for all $\mu \neq 0$. Our measurements instead sample valid schedule inputs. They therefore restrict K to the image of the key schedule. Write $\overline{\mathrm{Cr}}_{\mathrm{samp}}$ for the correlation averaged over those inputs. The statistic ρ of Subsection 6.3 estimates $\overline{\mathrm{Cr}}^2_{\mathrm{samp}}/\mathrm{RMS}^2$. For the SIMON cases and for a fixed-tweak run, the numerator is the squared master-key average. Consequently, ρ need not equal c_0^2/RMS^2. Restriction to the key-schedule image can give a nonzero round-key character a nonzero mean and can destroy orthogonality between distinct characters. In the extreme case, a nonzero character becomes constant on that image.

To quantify dependence under the actual schedule, let $k \in \mathbb{F}_2^\kappa$ be a uniform master key of length κ, and write $f(k) = \mathrm{Cr}(\mathsf{KS}(k))$, where KS is the key schedule and any tweak is fixed. Write its Walsh expansion as $f(k) = \sum_{\nu \in \mathbb{F}_2^\kappa} a_\nu (-1)^{\nu \cdot k}$, so $a_0 = \mathbb{E}_k f(k)$ is the master-key mean. Parseval's identity [O'D14, Proposition 1.13] yields

$$\mathrm{Var}_k(f) = \sum_{\nu \neq 0} a_\nu^2, \qquad \frac{\max_{\nu \neq 0} |a_\nu|}{|a_0|} \leq \sqrt{\frac{1 - \rho_{\mathrm{key}}}{\rho_{\mathrm{key}}}}, \quad \rho_{\mathrm{key}} = \frac{a_0^2}{\mathbb{E}_k f(k)^2}, \tag{7}$$

where the ratio requires $a_0 \neq 0$, and the measured ρ estimates ρ_{key}. Thus $\rho_{\mathrm{key}} > 1/2$ suffices for the constant coefficient to be strictly larger in magnitude than every nonconstant coefficient. Such dominance is weaker than negligible relative variation, which requires $\sum_{\nu \neq 0} a_\nu^2 \ll a_0^2$. Even this stronger concentration of Fourier energy permits rare keys with much smaller correlation.

Decompositions of this form come from the geometric approach to cryptanalysis. Beyne described linear cryptanalysis in a framework where fixed-key correlations are exact and are obtained by multiplying transition matrices written in a chosen basis [Bey21]. Beyne and Rijmen applied the same framework to differentials [BR22]. There the probability of a differential under a fixed key splits into contributions called quasidifferential trails, and each contribution carries a sign of the form $(-1)^{\mu \cdot K}$. Those masks μ are the key masks of Definition 3. Hu, Zhang, Chang, Zhang, Wang, and Peyrin later allowed different parts of a cipher to use different bases [HZC+25]. A DL distinguisher needs this mix-basis form, because its upper part propagates a difference while its lower part propagates a mask.

Subsection 8.2 describes the model that these bases give, and shows where a round key enters a trail.

8.2 Locating the Responsible Key Bits

Direct enumeration cannot find the large coefficients of Equation 6. For SIMON-32 over 13 rounds the vector K has 208 bits, so the expansion has 2^{208} terms. We therefore used a procedure with three steps. A search over trails produces a short list of candidate key bits. An exact measurement evaluates the bits on that list. A third step checks the answer on keys drawn from the real key schedule. This subsection describes the three steps for SIMON and explains why none of them can be skipped.

How the model describes one trail. The analysis follows a *pair* of messages through the cipher, so it tracks two components for every state word. The first component is

the *value mask*, which selects a parity of the value of that word. The second component describes how the two messages of the pair differ. Two bases are available, and the choice of basis decides what the second component means.

Table 10: State-word components in the two bases.

Basis	**First component**	**Second component**
Difference basis	Mask on the value	A difference, that is, the exact pattern of bits in which the two messages differ
Mask basis	Mask on the value	A mask that selects which bits of the difference the trail follows

The upper part of the cipher uses the difference basis and the lower part uses the mask basis, as Table 10 states. One *basis switch* sits at a chosen round boundary between them. These two bases are the quasidifferential basis of Beyne and Rijmen [BR22] and the linear basis of Beyne [Bey21], combined in the way Hu et al. describe [HZC+25]. The position of the switch is a modelling choice. It changes which trails the model enumerates, but not the decomposition of Equation 6, and our implementation tests that invariance.

Where the key enters. XORing a round key into a state word changes neither the difference between the two messages nor a mask on that difference. It changes only the value. A trail is therefore multiplied by $(-1)^{\nu \cdot k_i}$, where ν is the value mask of the word that k_i is added to. Collecting these masks over all rounds produces the trail key mask $\mu = (\mu_0, \ldots, \mu_{r-1})$ in the sense of Definition 3. In SIMON the round key is added to the new left word, so μ_i can be read directly from the value mask of the left word at the next round boundary.

Grouping trails by their key mask produces the coefficients of Equation 6. The coefficient c_μ is the sum of the correlations of all trails whose key mask is μ, and the trails with an all-zero key mask form c_0. A trail can depend on the key only when it activates a value mask, which provides a concrete search target.

Step 1: search for the lowest-weight key-dependent trails. Only the AND gates of the SIMON round function are nonlinear, and only they contribute to the trail weight. The model therefore encodes three kinds of operation. It encodes the copy and the three-input XOR of the round function as linear constraints over $\mathbb{F}_2$, at zero weight. It encodes each AND gate as a one-hot choice among the entries of its exact local transition matrix and assigns weight $-\log_2 |c|$ to an entry of correlation c. It encodes the basis switch as a free pairing of the difference with the difference mask, whose only effect is a sign.

The round-key mask bits are variables of this model, which is what makes key dependence visible to the search. Requiring at least one of them to be active and minimizing the total weight returns the lowest-weight key-dependent trail. Enumerating every trail within a small weight window above that optimum, then grouping the results by key mask, produces the short list.

A window of a few weight units captures only part of the trails that build up the correlation, so its signed sums are not bounds on the true coefficients. We read this output for which key masks appear, in what order, and with which signs, and not for the sizes of its numbers.

A key mask on this list is a candidate, not a conclusion. The coefficient c_μ sums the correlations of every trail whose key mask is μ, and trails sharing a mask can cancel. A mask can therefore appear in the search while its coefficient is zero, so the key bits it names do not affect the correlation at all. Hu et al. report this behaviour for SIMECK, where trails with nonzero value masks exist but sum to zero, leaving the correlation exactly

key-independent [HZC+25]. The reverse reading is safe: if the search proves that no key-dependent trail exists, then every trail has an all-zero key mask, so $c_\mu = 0$ for all $\mu \neq 0$ and the correlation is key-independent. Step 3 is what decides a positive claim.

Step 2: measure the listed bits exactly. Take the m key bits from the short list and fix every other round-key bit. Evaluate the full-codebook correlation at all 2^m settings of the chosen bits, which produces 2^m exact numbers. One WHT of length 2^m, normalized by 2^{-m}, applied to those numbers returns one coefficient for each subset of the chosen bits. The transform library serves both roles here: the same primitive that turns a codebook into correlations also turns those correlations into their expansion over the key bits.

Cost decides how the two steps divide the work. Each full-codebook evaluation of SIMON-32 took about nine seconds on ten threads, and this step doubles in cost with every added bit. That is why step 1 has to come first.

The returned numbers must be read as aggregates. The transform separates the chosen bits from each other exactly, but each bit held fixed contributes a fixed sign that the transform cannot remove. Every returned value is therefore the sum of all coefficients of Equation 6 whose key mask agrees with it on the chosen bits. In particular, the value returned for the zero mask is not the global coefficient c_0.

Step 3: confirm on keys from the real key schedule. Steps 1 and 2 treat the round keys as independent, which the key schedule does not. For each sampled master key, step 3 runs the real key schedule, reads the identified bits, evaluates the exact full-codebook correlation, and groups the keys by those bits. Two properties then decide whether the answer is right. The groups must lie far apart, and the spread inside each group must be small. A small spread inside the sampled groups also constrains empirically what step 2 could not see, because a comparable dependence on some other round key would show up as spread within a group. This step can also reject a candidate. When the groups do not separate, the key bits proposed by step 1 do not explain the behaviour, regardless of their trail weights. The measurement uses master keys that neither earlier step has seen, so it is independent of both the search and its weight model.

For affine key schedules, the selected round-key bits are affine master-key parities. Their linear parts specify the groups; constant offsets only relabel them. The following identity relates the variance within these groups to the Fourier coefficients not explained by the selected parities.

Proposition 1 (Fourier energy left outside the identified parities)**.** *Let $\nu_1, \ldots, \nu_d$ be linearly independent master-key masks, let S be their linear span, and let $g(k)$ be the mean of f over all keys sharing the parities $(\nu_i \cdot k)_{i=1}^{d}$. Then*

$$g(k) = \sum_{\nu \in S} a_\nu (-1)^{\nu \cdot k}, \qquad \mathbb{E}_k\big(f(k) - g(k)\big)^2 = \sum_{\nu \notin S} a_\nu^2.$$

Proof. A character with mask in S is constant inside each group. Every other character averages to zero there, since translation by a vector orthogonal to S can reverse its sign while preserving the group. Applying this average to the Walsh expansion proves the first equality; Parseval proves the second. □

The within-group variance therefore measures the Fourier energy outside S, including contributions absent from the trail enumeration. Its empirical estimate quantifies the key dependence that the proposed grouping does not explain. The coefficients in S can likewise be estimated by a normalized WHT of the mean correlations obtained from uniform master-key samples within each parity class. Unlike the restricted round-key experiment of step 2, this estimation respects the key schedule and averages over unrelated key bits. The number of classes is 2^d, determined by the rank of the selected parities.

8.3 The 13-Round SIMON-32 Distinguisher

We applied the procedure of Subsection 8.2 to the DL distinguisher of Figure 7. The model for this window has about 8,600 variables. The minimum trail weight is 28, and the minimum key-dependent trail weight is 29. Enumerating every trail within 2 weight units of the optimum returned 39,056 key-dependent trails. Table 11 lists the leading key masks.

Table 11: Leading key masks of the 13-round SIMON-32 model.

Key mask	Trails	Weight	Summed correlation
k_{11}: 0x8000	6144	29	$+2^{-18.83}$
k_{11}: 0x0040	6144	29	$-2^{-18.83}$
k_{11}: 0x8040	480	30	$+2^{-22.68}$
k_7: 0x8000	4736	29	$+2^{-23.00}$
k_7: 0x0040	4992	29	0
k_6: 0x2000	6112	29	0
29 further key masks, each summing to 0			

Two masks lead the list: bit 15 and bit 6 of the round key k_{11}. They rest on 6,144 trails each and carry equal magnitude with opposite signs. The joint mask on both bits follows one weight level later. The largest mask on any other round key is smaller by a factor of about 18, and the remaining key masks in the window sum to zero. Table 12 prints the lowest-weight trail behind the first of these two masks.

Table 12: A lowest-weight trail for the key mask k_{11}:0x8000.

Boundary	x value	x second	y value	y second	Basis	
0	0x0000	0x0001	0x0000	0x0004	difference	input difference
1	0x0000	0x0000	0x0000	0x0001	difference	
2	0x0000	0x0001	0x0000	0x0000	difference	
3	0x0000	0x0004	0x0000	0x0001	difference	
4	0x0000	0x0011	0x0000	0x0004	difference	
5	0x0000	0x0040	0x0000	0x0011	difference	
6	0x0000	0x4111	0x0000	0x0040	difference	basis switch
7	0x0000	0x0088	0x0000	0x0020	mask	
8	0x0000	0x0020	0x0000	0x0080	mask	
9	0x0000	0x0080	0x0000	0x0000	mask	
10	0x0000	0x0000	0x0000	0x0080	mask	
11	0x0000	0x0080	0x8000	0x8020	mask	
12	0x8000	0x8020	0x0000	0x0088	mask	key mask of k_{11}
13	0x0000	0x0088	0x0000	0x8002	mask	output mask

Each row of Table 12 reports the two components of the left word x and of the right word y at one round boundary. Three rows show how to read the whole table.

Boundary 4 lies above the switch, so its second components are differences. They say that the two messages differ in the left word at bits 0 and 4, because 0x0011 has those bits set, and in the right word at bit 2. Both value masks are zero, so this round collects no key sign. The difference travels down the table as SIMON moves it: the new right word is the old left word, so the right second component at boundary 5 repeats the left second component at boundary 4.

Boundary 9 lies below the switch, so its second component is no longer a difference. The value 0x0080 is a mask, and it says that the trail follows bit 7 of the difference of the left word. The right word has mask 0x0000, so no bit of its difference is used. This is the change that the switch makes: above it a number *is* the difference, and below it a number *selects* which bits of the difference the trail uses.

Boundary 12 is the only row whose value mask is not zero. The mask 0x8000 is bit 15, and the key k_{11} is added to the left word that becomes boundary 12. This mask is therefore the key mask of round 11, and the trail carries the sign $(-1)^{k_{11}[15]}$. Every other row has value mask 0x0000, which is why this trail depends on one key bit only. The second leading trail agrees with this one up to boundary 10 and then places the value mask on bit 6 instead of bit 15.

The last round explains why these two bits, and no others, lead the list. Restricted to the left word, the output mask 0x00888002 is 0x0088, so its bit 7 is active. In SIMON, the AND gate that produces output bit 7 reads state bits $7-1=6$ and $7-8=15$, following the rotation amounts of Subsection E.1. A value mask can sit on either input of that AND gate, and these two choices are exactly the two leading trails. The word those inputs belong to is the output of round 11, which is the word that k_{11} is added to. The two choices enter the correlation with opposite signs, so their coefficients have opposite signs. The same output mask also has bit 3 active, whose AND gate reads state bits 2 and 11. The measurement below puts the coefficients on $k_{11}[2]$ and $k_{11}[11]$ at $2^{-16.6}$ and below, so the branch through bit 7 dominates.

Step 2 first screened the four bits $k_{11}[15]$, $k_{11}[6]$, $k_{11}[11]$, and $k_{11}[2]$. The first two dominated, so Table 13 reports a separate exact two-bit measurement with every other round-key bit fixed. Writing $a = k_{11}[15]$ and $b = k_{11}[6]$, its left half lists the exact full-codebook correlation at each of their four settings, and its right half lists the coefficients returned by one WHT of those four numbers.

Table 13: Exact two-bit correlations and their Walsh coefficients.

a	b	**Exact correlation**	**Key-mask pattern**	**Coefficient**
0	0	$2^{-11.39}$	neither bit	$+2^{-11.77}$
1	0	$2^{-13.94}$	a only	$+2^{-13.59}$
0	1	$2^{-11.43}$	b only	$-2^{-13.84}$
1	1	$2^{-11.49}$	both bits	$+2^{-13.73}$

The three sign-carrying coefficients have comparable size, and the one on b is negative while the other two are positive. That is the arrangement the trail search predicted. It also explains the measured correlations. When $a = 1$ and $b = 0$, the three sign-carrying terms all oppose the constant term, and the correlation drops from about $2^{-11.4}$ to $2^{-13.9}$. For the other three settings they do not all oppose it, and the correlation stays near $2^{-11.4}$. The coefficients on $k_{11}[11]$ and $k_{11}[2]$ came out at $2^{-16.6}$ and below, at least 8 times smaller than the 2 leading ones. As Subsection 8.2 explains, these four values are aggregates over the 206 bits held fixed, so we use them for their relative sizes and signs.

Step 3 removed that reservation. Under the SIMON-32 key schedule, each of the two bits is an affine function of 29 bits of the 64-bit master key, so neither is visible from the master key by inspection. We computed both bits for the 10,000 confirmation keys that produced Figure 7 and grouped the measured correlations accordingly. Table 14 reports the four groups.

Table 14: Key groups for the 13-round SIMON-32 distinguisher.

$k_{11}[15]$	$k_{11}[6]$	Keys	Share (%)	RMS
0	0	2480	24.8	$2^{-11.77}$
0	1	2563	25.6	$2^{-11.78}$
1	0	2489	24.9	$2^{-14.64}$
1	1	2468	24.7	$2^{-11.78}$

The four groups have nearly equal size, as two balanced parity bits predict. Three of them agree to within 0.01 in the exponent. The fourth is weaker by a factor of 7.2, which

matches the drop that Table 13 predicts. The four-group model explains 93.1% of the signed variance in the 10,000 measurements. The remaining 6.9% lies within the groups and estimates the unexplained Fourier energy of Proposition 1. Although this residual is small relative to the total variance, the weak group's standard deviation is 67.1% of the magnitude of its signed mean, compared with about 11% in each strong group. The two identified bits thus explain the separation between the peaks of Figure 7, but substantial relative variation remains within the weak class. The prediction came from the trail search and the key-bit measurement, and the grouping confirmed it on keys that neither step had seen.

8.4 An 11-Round SIMON-32 Distinguisher from the Literature

The previous case used a pair of our own. We also applied the procedure to a published distinguisher, the 11-round DL pair of Table 5 with input difference `0x00080022` and output mask `0x00400010` [CXZQ25]. Our confirmation RMS for that distinguisher is $2^{-7.91}$, which reproduces the published value. Figure 8 shows its distribution over 10,000 independently sampled master keys. It has three visible peaks rather than two, and they span only 0.4 bits.

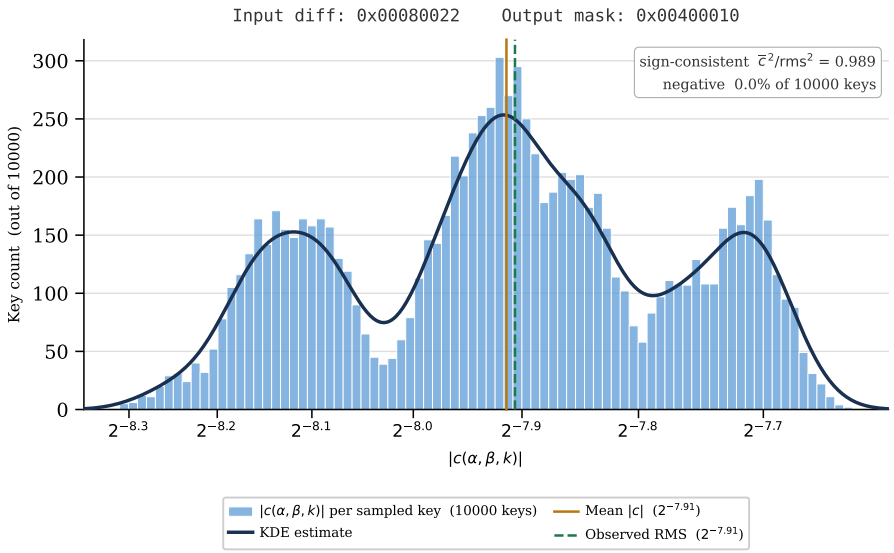


Figure 8: The published 11-round DL distinguisher separates into three visible peaks that together span 0.4 bits, so its RMS remains a faithful summary.

Step 1 built a model with 7,312 variables and 2,881 constraints. The minimum key-dependent trail weight is 21, and enumerating every trail within 2 weight units returned 3,424 trails over 15 distinct key masks. Table 15 lists the leading entries.

Step 3 then measured the candidates on the 10,000 confirmation keys. The confirmation measurement contradicted the in-window ranking in two places. The mask k_7: `0x0020` is tied for the minimum weight and largest trail count, yet grouping the keys by $k_7[5]$ does not separate the measured correlations. The joint mask on $k_1[12]$ and $k_3[12]$ sums to 0 inside the enumerated window, yet it turns out to carry the largest coefficient of all. Both discrepancies have the same source, stated in Subsection 8.2: an in-window sum aggregates the trails that the enumeration reached, so it orders candidates but does not measure them. Table 16 groups the keys by the two masks that survive the measurement.

Their measured coefficients are $+2^{-11.560}$ on $k_1[12] \oplus k_3[12]$ and $+2^{-12.048}$ on $k_7[14]$,

Table 15: Leading key masks of the 11-round SIMON-32 model.

Key mask	**Trails**	**Weight**	**Summed correlation**
k_7: 0x4000	928	21	$+2^{-13.68}$
k_7: 0x0020	928	21	$+2^{-14.00}$
k_7: 0x4020	96	22	$-2^{-17.00}$
k_1: 0x1000, k_3: 0x1000	16	22	0
k_5: 0x0400	928	21	0
10 further key masks, each summing to 0			

Table 16: Key groups for the published 11-round SIMON-32 distinguisher.

$k_1[12] \oplus k_3[12]$	$k_7[14]$	Keys	Share (%)	RMS
0	0	2497	25.0	$2^{-7.73}$
0	1	2478	24.8	$2^{-7.88}$
1	0	2483	24.8	$2^{-7.94}$
1	1	2542	25.4	$2^{-8.13}$

against a master-key mean of $2^{-7.914}$. The squared character projections account for 59.9% and 30.4% of the variance across keys, and their interaction for only 0.08%. Because the four sampled groups are not exactly balanced, the direct four-group decomposition yields the nearby total of 91.1%. The same grouping explains 91.3% on the first 5,000 keys and 90.9% on the disjoint second 5,000, so the result is not driven by a small subset of the sample. The unequal coefficients explain why the distribution has three visible peaks rather than four. The two coefficients are close but not equal, so the middle two groups land at $2^{-7.88}$ and $2^{-7.94}$ and merge into one visible peak, while the outer groups stay apart.

The outcome differs from the 13-round case in degree, not in kind. The key dependence is real and concentrated on two masks, both of which occur in the model's candidate list even though their in-window ranking is wrong. The measured dependence is also small: the strongest group exceeds the weakest by a factor of 1.32, every one of the 10,000 keys keeps the same sign, and the sign statistic ρ is 0.99. The weakest group's RMS is 0.22 bits below the overall RMS, so sizing from the overall value understates its group-average data by a factor of about 1.36, rather than the factor of about 40 seen at 13 rounds. Structure in the distribution does not by itself make a summary misleading, and step 3 is what settles the difference.

8.5 The 12-Round SIMON-32 Distinguisher

The 12-round pair provides a contrasting regime one round below the 13-round distinguisher. The 12-round SIMON-32 distinguisher of Table 5 has input difference 0x40000001 and output mask 0x00080022. Its confirmed RMS is $2^{-8.95}$ over 10,000 keys, its correlation is positive on every one of them, and its sign statistic is $\rho = 0.993$. Almost all of its power therefore survives averaging over master keys, but this does not make the correlation key-independent.

Table 17 lists every key mask that appears within 2 weight units of the optimum.

Within that window the seven key-dependent groups do cancel exactly, while the 2,912 key-independent trails carry the entire enumerated correlation. They are partial sums, however, not full coefficients. Table 18 shows what happens when the window is widened.

The mask on bit 13 of k_7 changes from zero to $+2^{-19.00}$, $+2^{-18.00}$, and $+2^{-16.83}$ in the three wider windows. The initial cancellation therefore does not establish a zero coefficient; it identifies a candidate whose contribution begins beyond that window. We tested that candidate directly on the 10,000 confirmation keys.

Table 17: Key-mask partial sums within 2 weight units of the 12-round optimum.

Key mask	Trails	Weight	Summed correlation
(key-independent)	2912	24	$+2^{-14.73}$
k_6: 0x0800	160	25	0
k_7: 0x0010	128	25	0
k_7: 0x2000	128	25	0
k_7: 0x2010	384	26	0
k_6: 0x0800, k_7: 0x2000	256	26	0
k_6: 0x0800, k_7: 0x0010	256	26	0
k_4: 0x0080, k_5: 0x0200	80	26	0

Table 18: Partial correlation sums for wider 12-round trail windows.

Window	Max weight	Trails	Total sum	k_7: 0x2000	Time (s)
2	26	4304	$+2^{-14.73}$	0	3.7
3	27	22 384	$+2^{-14.44}$	$+2^{-19.00}$	30.3
4	28	119 312	$+2^{-12.77}$	$+2^{-18.00}$	154.3
5	29	523 392	$+2^{-12.41}$	$+2^{-16.83}$	983.1

Table 19: Key groups for the 12-round SIMON-32 distinguisher. The variance between these groups is 66.9% of the total correlation variance.

$k_7[13]$	Keys	Share (%)	RMS
0	5029	50.3	$2^{-8.86}$
1	4971	49.7	$2^{-9.06}$

The corresponding measured character coefficient on the master-key space is $+2^{-12.86}$, and the two group RMS values differ by a factor of 1.14. The model therefore locates most of the modest key dependence, while the residual 33.1% of the variance shows that this one parity is not a complete decomposition. The explained share is 67.2% on the first 5,000 keys and 66.6% on the disjoint second 5,000. The standard deviation divided by the signed mean, that is, the coefficient of variation, is 8.17% across all keys. The largest relative deviation from the signed mean is 20.5%, and the sign never changes.

The total trail sum in Table 18 remains more than 3 bits below the measured magnitude, so enumeration is not used to estimate the correlation. Exact magnitudes and the grouping above come from the full-codebook measurements of Subsection 6.3; the trail model supplies the candidate parity.

Taken with Subsection 8.3, these two distinguishers show both regimes on adjacent round counts of one cipher. At 13 rounds two masks lead all other masks by a factor of about 18, and two key bits split the keys by a factor of 7.2. At 12 rounds one predicted parity explains most of a much narrower split, only a factor of 1.14, so the RMS remains representative even though the correlation is not key-independent.

8.6 The 84-Round KATAN-32 Distinguisher

The two SIMON-32 cases share a round function built from AND, XOR and rotation. KATAN-32 is built differently, from two coupled nonlinear feedback shift registers with an irregular update rule, and it produces only two state bits per round. Its round function is nonetheless AND, XOR and copy, so the local relations of Subsection 8.2 apply unchanged and no new gate machinery is needed. A window of 84 rounds produces a model of 27,324 variables and 14,080 constraints, comparable to 13 rounds of SIMON-32, because the shift that carries most of the state is a relabelling and has zero weight. We validated the

new wiring independently before using it here. Its encryption agrees with three known-answer cases from the C implementation, and on a five-round instance the complete trail decomposition agrees with direct evaluation of the DL definition over the exact causal input cone for every possible basis-switch boundary.

We applied the procedure to the 84-round distinguisher of Table 6, with input difference `0x00008010` and output mask `0x20044000` from start round zero. Its measured behaviour is the most sign-consistent in this paper. Over 1,000 keys it has RMS $2^{-12.18}$ and $\rho = 0.990$, and its correlation is negative on every single key.

The leading orders reproduce that sign-consistent behaviour. The minimum trail weight is 25, attained by a trail with no key mask. Enumerating exhaustively through weight 28 returns 83,073 trails, and every one of them is key-independent. Their correlations sum to $-2^{-13.59}$, so the truncated model predicts a negative correlation, which is what all 1,000 measured keys show. The enumerated magnitude falls 1.4 bits short of the measured $2^{-12.18}$; we use this comparison only as a consistency check, not as an estimate of the omitted trails.

Key dependence first becomes possible at weight 29, 4 units above the optimum. Table 20 lists the key-dependent trails at that weight, enumerated exhaustively.

Table 20: Key-dependent trail sums at weight 29 for the 84-round KATAN-32 distinguisher.

Key mask	Trails	Weight	Summed correlation
k_{80}, k_{97}	8192	29	0
k_{84}	1536	29	0

Both masks sum to exactly zero. Through weight 29 the correlation of this distinguisher is therefore entirely key-independent: no trail below that weight carries a key mask at all, and every trail that does carry one cancels. The three round-key bits involved are the first key of round 40, the first key of round 42 and the second key of round 48, so they sit near the middle of the window.

This is the same leading-order cancellation pattern seen in the narrow window of Subsection 8.5, on a cipher whose construction has nothing in common with SIMON beyond its use of AND gates. Together the two show that the enumerated leading contribution of a key mask returned by the search can vanish on unrelated designs, which is why the search names candidates and step 3 decides them.

One limit is worth stating. The enumeration is exhaustive only through weight 29, and ρ is 0.990 rather than exactly 1, so some key dependence must survive at higher weight. What the model establishes is that the leading key-dependent order cancels, not that the distinguisher is exactly key-independent. Infeasibility of the key-dependent model would suffice to establish exact independence, but complete cancellation of all nonzero-mask coefficients could establish it even for a feasible model.

8.7 Scope and Limits of the Procedure

The four cases above illustrate several outcomes of the procedure, and a short list of effective key bits is only one of them.

The procedure names such a list only when the key dependence is concentrated on a few masks. This concentration occurs in all three SIMON-32 cases, although the practical separation differs sharply among them. The 13-round SIMON-32 distinguisher of Subsection 8.3 has two masks leading all other masks by a factor of about 18, and its groups differ by a factor of 7.2. The published 11-round DL distinguisher of Subsection 8.4 also has two responsible masks, but its groups differ by a factor of only 1.32. At 12 rounds one predicted parity explains 66.9% of the variance, but its two groups differ by only a factor of 1.14. A correct list of responsible key bits does not by itself imply that the

RMS is misleading. Only the separation measured in step 3 shows whether one reported correlation describes every key. The list and the separation are properties of a distinguisher, not of the method. For the 84-round KATAN-32 distinguisher the enumerated leading key-dependent contributions cancel, so no short list emerges at that order; its correlation also keeps one sign on every measured key. The BEANIE result illustrates another outcome. We ran the same steps on the 3+2 BEANIE distinguisher of Table 7, whose sign varies with the key. Here the search cost is the sum of the base-two logarithms of the denominators of the nonzero local coefficients in lowest terms; it upper-bounds the exact trail weight. The saved solution pool within one cost unit of the optimum contains 981 trails. Of these trails, 690 are key-dependent and occupy 29 distinct masks. The solver did not prove that this pool exhausts the cost window. We mapped the 29 masks through the five-round tweakey schedule onto the two suffixes. We then projected the resulting parities onto the 1,000 fixed-tweak measurements of Subsection 8.8. We measured multivariate fit with the coefficient of determination R^2, the fraction of variance explained relative to the constant-mean predictor. The best single candidate explains only 1.64% of the correlation variance. The 29-candidate model has an in-sample R^2 of 8.12%. The best individual bit among the 224 derived round-key bits explains 0.56%. Repeated 10-fold cross-validation gave a median out-of-sample R^2 of 1.84% over 20 fixed-seed repetitions. The values ranged from 1.04% to 2.90%. The candidate family therefore provides weak predictive information, but leaves most of the key-dependent variation unexplained. The incomplete pool does not exclude missing masks within the same cost window or an explanation beyond the tested search budget. These cases show that a short list is a measured outcome, not an assumption of the method. Step 3 distinguishes the cases.

We also tried a cheaper substitute for the procedure and report it because it fails. Solving the model twice, once restricted to key-independent trails and once to key-dependent trails, produces two optimal search costs in seconds. Their difference measures the additional cost of a key-dependent trail, so a small difference suggests that the correlation should move with the key. It does not. The minimum key-dependent trail weight is one unit above the overall minimum for both the 12-round and the 13-round SIMON-32 pairs of Table 5. Nevertheless, the 12-round correlation has an 8.17% coefficient of variation and never changes sign over 10,000 keys, while the 13-round correlation splits by a factor of 7.2. The two BEANIE pairs of Table 7 also disagree. The 3+2 pair has a search-cost difference of zero and nearly equal frequencies of positive and negative signs, whereas the 2+3 pair has a search-cost difference of one and a 2.62% coefficient of variation. A one-unit gap therefore accompanies both stable and unstable correlations on two ciphers. The reason is that this shortcut compares the single lowest-cost trail in each class, while a coefficient sums over very many trails, and a class with more trails can easily overturn a one-unit advantage. We therefore do not use this gap to rank or to discard candidates.

The procedure itself is not restricted to 32-bit blocks, although exact full-codebook confirmation is.

Appendix D gives a partial schedule-aware analysis of a six-round SIMON48/96 middle. The analysis combines exact cancellation of the enumerated leading-order terms with a statistical bound establishing that the constant master-key coefficient is largest. A stronger structural account requires control of the unenumerated tail.

8.8 Sign Behaviour Across the Five Targets

The confirmation runs record signed correlations, while the distribution figures plot $|c|$, so the sign is a separate observable, summarized by the statistic ρ of Subsection 6.3.

Every linear distinguisher we confirmed behaves the same way. Across all five ciphers ρ is at most 0.06 and the proportion of negative values lies between 43% and 52%. That is the expected key dependence of a linear approximation's sign, and it is the reason the squared correlation, rather than the correlation, composes in DL analysis.

Most confirmed DL distinguishers retain a substantial signed mean, with ρ from 0.62 to 1.00. The 15-round SIMON-32 pair has $\rho = 0.271$, with 27.28% negative values and one zero among 10,000 keys. Its sign therefore varies substantially, but its positive-sign imbalance provides the full-codebook distinguishing evidence described in Subsection 7.1.

The 3+2 BEANIE DL result shows still greater cancellation in the signed mean. Its RMS is $2^{-8.46}$, above the $2^{-15.50}$ DL random-permutation reference, yet $\rho = 0.002$ with 50.4% negative values. This contrasts with its 2+3 counterpart, whose ρ is 0.999, and with the partial cancellation of the 15-round SIMON-32 pair. The joint key-and-tweak average is close to zero while the varying component is large; this observation alone says nothing about the independent-round-key coefficient c_0.

BEANIE is tweakable, and the variable-tweak run samples a fresh tweak pair with every key, so the observed spread could in principle come from the tweaks rather than from the key. We therefore repeated the measurement with both tweaks held fixed and only the key redrawn, over the same 1,000 keys. The fixed-tweak run has RMS $2^{-8.52}$, $\rho < 0.001$, and 49.7% negative values; see Figure 9. These measurements show that key variation alone is sufficient to change the sign. This is compatible with the tweakey schedule, which mixes the key into the tweak state before every nonlinear update and derives the round keys from the resulting state. Explaining which key bits are responsible needs a mix-basis model of the U-shape construction, which composes an encryption and a decryption under two tweaks. We built that model and applied the procedure of Subsection 8.2 to this distinguisher. The quantitative projection reported in Subsection 8.7 finds weak predictive information in the 29 collected candidate masks, while most variation remains unexplained; individual derived round-key bits explain still less. The full-codebook experiment establishes the reported magnitude and sign statistics for the sampled keys. The structural conclusion remains limited to the trail budget and candidate family that we tested.

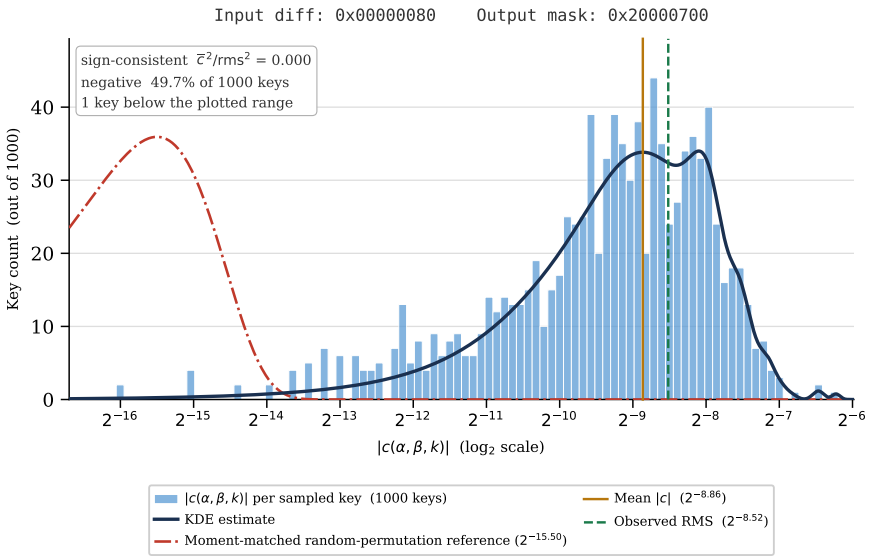


Figure 9: Holding both BEANIE tweaks fixed and redrawing only the key produces nearly equal frequencies of positive and negative signs in the 3+2 DL correlation.

8.9 Consequences for Reporting a Distinguisher

Three practical consequences follow from the measurements above.

The first concerns the shape of a per-key distribution. Subsection 6.2 derives the

correlation laws of a random permutation, and those laws approach a normal law as the block size grows. They describe the ideal object against which a distinguisher is compared, not the distinguisher itself. The distribution in Figure 7 has two peaks, so no unimodal law fits it, whether normal or half-normal. Its weak tail reaches $2^{-29.00}$ and is visible only on the logarithmic axes of Figure 7b, which is why we show both views. A cryptanalyst who assumes a normal per-key distribution when sizing an experiment describes three quarters of the keys and misses the rest. Several peaks are not particular to our own pairs, because the published 11-round distinguisher of Figure 8 has three.

The second concerns the RMS. Strong keys dominate a mean square, so the RMS of $2^{-11.98}$ overstates the weak-key correlation of $2^{-14.64}$ by a factor of 6.3. The data needed for a DL distinguisher grows with the inverse square of the correlation. The weak group's RMS therefore implies about 40 times the data budget obtained from the overall RMS. This is a comparison of group summaries, not a per-key guarantee. The mean of the magnitudes shares this weakness, because it also averages the two groups into one value. Neither statistic is wrong, and both remain the right choice for ranking candidates. Neither one, on its own, states which keys are weak.

Data complexity must therefore be interpreted relative to the fraction of keys for which the attack is intended to succeed. Under the inverse-square model, a sampled key with signed correlation d_i requires $M_i = (\mathrm{RMS}_{\mathrm{conf}}/|d_i|)^2$ times the data budget derived from the overall RMS, with $M_i = \infty$ when $d_i = 0$. For $0 < p \leq 1$, the empirical p-quantile of these factors is the smallest multiplier sufficient for at least a fraction p of the sample under this model. The median factor within the weak class is 56, whereas covering 95% or 99% of all 10,000 sampled keys requires factors of approximately 246 or 5,905, respectively. These are empirical coverage estimates, not confidence bounds for the corresponding fractions of the full key space. In general, the reciprocal of a group's mean squared correlation differs from its mean per-key inverse-square cost.

The third concerns what to report alongside the RMS. We report ρ and the full per-sample distribution with every result, and recommend also stating correlation quantiles or data multipliers at the intended key coverage. The statistic ρ estimates $\overline{\mathrm{Cr}}^2_{\mathrm{samp}}/\mathrm{RMS}^2$, so it separates the signed mean over the sampled schedule inputs from their varying part at no extra measurement cost. For ordinary ciphers and fixed-tweak runs, these quantities are the master-key mean and the key-varying part. The distribution figure then shows the shape that no single number carries. When the dependence is concentrated on a few masks, the procedure of Subsection 8.2 identifies the responsible key bits. Otherwise, the procedure measures what the tested trail budget and candidate family explain, as in the BEANIE example.

9 Conclusion and Future Work

Linear cryptanalysis, DL cryptanalysis, the DLCT, and recent spectral methods all transform large integer or real arrays. No available library combines integer and floating-point transforms, Boolean conventions, batching, CPU and GPU backends, and an embeddable C API. `LibFWHT` provides that interface in portable C99, with scalar, SIMD, multi-core, and GPU paths. On one CPU core it is 2.5 to 3.3 times faster than `FFTW`'s valid indirect route, and 2.00 times faster than `sboxU` on a Boolean spectrum of 2^{29} entries. Its CUDA backend reaches 1.82 TOps/s on an NVIDIA H200, where the copies to and from the device cost about 20 times the kernel, so the gain comes from keeping data resident for a whole search rather than from the kernel alone. The same transform also serves key recovery [CSQ07], S-box analysis [Car21, PAB+26], and recent spectral searches [DDK+23, HNW26], which repeat one transform many times on arrays that outgrow a single core.

The case studies supply results that were not previously available. For KATAN-32, BEANIE, KeeLoq, and RC5-16 we report the first exact DL distinguishers, with correlations

measured over the complete codebook and confirmed on independently sampled keys. For SIMON-32 the confirmed correlations improve on published results at several matched round counts, while a 1,000-key fixed-input sweep rediscovers the published 15-round DL pair. For KeeLoq, ciphertext-side NLF inputs refine the correlation phase of Bogdanov's basic linear slide attack [Bog07], reducing its modeled complexity from $2^{50.6}$ to $2^{48.82}$ encryptions while leaving the later 2^{37} attack unchanged.

Measuring a distinguisher on many keys makes its key dependence visible, and Section 8 analyzes it with the geometric approach. The dependence may concentrate on a few round-key bits, which the procedure names and a fresh sample of keys confirms. Its leading key-dependent orders may also cancel, a fact that is consistent with sign stability but does not by itself prove exact key independence. In another outcome, the tested candidate family may explain only a small part of the measured spread, in which case the tested trail budget does not justify a short list of responsible bits. A key mask that the search returns is therefore a candidate, and only the measurement decides it.

The practical consequence is that one number is not enough. A correlation measured over keys need not follow a normal law, and a root-mean-square value is dominated by the strong keys, so it can understate the data an attack needs on a substantial fraction of them. Even after identifying a weak class, its internal tail matters for the fraction of keys an attack can cover. We report the sign statistic and the full per-sample distribution with every result, and recommend stating the intended key coverage when interpreting attack costs.

To extend these analyses to more candidates and keys, the most important next step is exact acceleration on double-precision matrix units. On the H200 used here, these units provide twice the double-precision throughput of the vector units, while double precision preserves the integer-valued Boolean spectra considered here exactly. Future work should determine whether mapping groups of Walsh–Hadamard stages to double-precision matrix operations yields a net speedup after data rearrangement and memory traffic.

Acknowledgments

This project was supported by ERC project 101097056 (SYMTRUST). The authors thank the developers of the open-source tools used for comparison and verification. Parts of the calculations for this publication were performed on the HPC cluster Elysium of the Ruhr University Bochum, subsidised by the DFG (INST 213/1055-1).

References

[AAA+15] Mohamed Ahmed Abdelraheem, Javad Alizadeh, Hoda A. Alkhzaimi, Mohammad Reza Aref, Nasour Bagheri, and Praveen Gauravaram. Improved linear cryptanalysis of reduced-round SIMON-32 and SIMON-48. In *Progress in Cryptology – INDOCRYPT 2015*, volume 9462 of *LNCS*, pages 153–179. Springer, 2015. doi:10.1007/978-3-319-26617-6_9.

[AK18] Ralph Ankele and Stefan Kölbl. Mind the gap - A closer look at the security of block ciphers against differential cryptanalysis. In Carlos Cid and Michael J. Jacobson Jr., editors, *SAC 2018*, volume 11349 of *LNCS*, pages 163–190. Springer, 2018. doi:10.1007/978-3-030-10970-7_8.

[BDKW19] Achiya Bar-On, Orr Dunkelman, Nathan Keller, and Ariel Weizman. DLCT: A new tool for differential-linear cryptanalysis. In *EUROCRYPT 2019*, volume 11476 of *LNCS*, pages 313–342. Springer, 2019. doi:10.1007/978-3-030-17653-2_11.

[Bey21] Tim Beyne. A geometric approach to linear cryptanalysis. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021*, LNCS, pages 36–66. Springer, 2021. doi:10.1007/978-3-030-92062-3_2.

[Bog07] Andrey Bogdanov. Linear slide attacks on the KeeLoq block cipher. In Dingyi Pei, Moti Yung, Dongdai Lin, and Chuankun Wu, editors, *Information Security and Cryptology*, volume 4990 of *LNCS*, pages 66–80. Springer, 2007. doi:10.1007/978-3-540-79499-8_7.

[BR22] Tim Beyne and Vincent Rijmen. Differential cryptanalysis in the fixed-key model. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022*, LNCS, pages 687–716. Springer, 2022. doi:10.1007/978-3-031-15982-4_23.

[BS90] Eli Biham and Adi Shamir. Differential cryptanalysis of DES-like cryptosystems. In Alfred Menezes and Scott A. Vanstone, editors, *CRYPTO 1990*, volume 537 of *LNCS*, pages 2–21. Springer, 1990. doi:10.1007/3-540-38424-3_1.

[BSS+15] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The SIMON and SPECK lightweight block ciphers. In *DAC 2015*, pages 175:1–175:6. ACM, 2015. doi:10.1145/2744769.2747946.

[Car21] Claude Carlet. *Boolean Functions for Cryptography and Coding Theory*. Cambridge University Press, 2021. doi:10.1017/9781108606806.

[CBB08] Nicolas T. Courtois, Gregory V. Bard, and Andrey Bogdanov. Periodic ciphers with small blocks and cryptanalysis of KeeLoq. *Tatra Mountains Mathematical Publications*, 41:167–188, 2008. URL: https://www.sav.sk/journals/uploads/0219142612cobabo.pdf.

[CDK09] Christophe De Cannière, Orr Dunkelman, and Miroslav Knezevic. KATAN and KTANTAN: A family of small and efficient hardware-oriented block ciphers. In Christophe Clavier and Kris Gaj, editors, *CHES 2009*, volume 5747 of *LNCS*, pages 272–288. Springer, 2009. doi:10.1007/978-3-642-04138-9_20.

[CSQ07] Baudoin Collard, François-Xavier Standaert, and Jean-Jacques Quisquater. Improving the time complexity of Matsui's linear cryptanalysis. In Kil-Hyun Nam and Gwangsoo Rhee, editors, *ICISC 2007*, volume 4817 of *LNCS*, pages 77–88. Springer, 2007. doi:10.1007/978-3-540-76788-6_7.

[CXZQ25] Siwei Chen, Zejun Xiang, Xiangyong Zeng, and Guangxue Qin. Enhancing the MILP/MIQCP-based automatic search for differential-linear distinguishers of IoT-friendly block ciphers Simon and Simeck. *IEEE Internet of Things Journal*, 12(5):5655–5671, 2025. doi:10.1109/JIOT.2024.3486965.

[CZ18] Yanqin Chen and Wenying Zhang. Differential-linear cryptanalysis of SIMON32/64. *International Journal of Embedded Systems*, 10(3):196, 2018. doi:10.1504/IJES.2018.091778.

[Dao23] Dao-AILab contributors. fast-hadamard-transform: Fast Hadamard Transform in CUDA with a PyTorch interface. https://github.com/Dao-AILab/fast-hadamard-transform, 2023. Accessed 2026-06-08.

[DDK+23] Itai Dinur, Orr Dunkelman, Nathan Keller, Eyal Ronen, and Adi Shamir. Efficient detection of high probability statistical properties of cryptosystems via surrogate differentiation. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023*, LNCS, pages 98–127. Springer, 2023. doi:10.1007/978-3-031-30634-1_4.

[DM98] Leonardo Dagum and Ramesh Menon. OpenMP: An industry-standard API for shared-memory programming. *IEEE Computational Science & Engineering*, 5(1):46–55, 1998. doi:10.1109/99.660313.

[DML24] DMLC Community. DLPack: Open in-memory tensor structure for sharing across frameworks. https://github.com/dmlc/dlpack, 2024. Accessed 2026-04-16.

[FA76] Bernard J. Fino and V. Ralph Algazi. Unified matrix treatment of the fast Walsh–Hadamard transform. *IEEE Transactions on Computers*, C-25(11):1142–1146, 1976. doi:10.1109/TC.1976.1674569.

[FAL15] FALCONN-LIB contributors. FFHT: Fast Fast Hadamard Transform. https://github.com/FALCONN-LIB/FFHT, 2015. Accessed 2026-06-08.

[FJ05] M. Frigo and S. G. Johnson. The design and implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005. doi:10.1109/JPROC.2004.840301.

[FJ20] Matteo Frigo and Steven G. Johnson. *FFTW User Manual*. FFTW Project, 2020. Version 3.3.10, dated 10 December 2020.

[GHM+25] Simon Gerhalter, Samir Hodžić, Marcel Medwed, Marcel Nageler, Artur Folwarczny, Ventzi Nikov, Jan Hoogerbrugge, Tobias Schneider, Gary McConville, and Maria Eichlseder. BEANIE – a 32-bit cipher for cryptographic mitigations against software attacks. Cryptology ePrint Archive, Paper 2025/2186, 2025. URL: https://eprint.iacr.org/2025/2186, doi:10.46586/tosc.v2025.i4.31-69.

[Goh19] Aron Gohr. Improving attacks on round-reduced speck32/64 using deep learning. Cryptology ePrint Archive, Paper 2019/037, 2019. URL: https://eprint.iacr.org/2019/037.

[Goo58] I. J. Good. The interaction algorithm and practical Fourier analysis. *Journal of the Royal Statistical Society, Series B (Methodological)*, 20(2):361–372, 1958. doi:10.1111/j.2517-6161.1958.tb00300.x.

[had26] hadipourh. KeeLoq reference implementation. https://github.com/hadipourh/KeeLoq, 2026. Accessed 2026-08-26.

[HDE24] Hosein Hadipour, Patrick Derbez, and Maria Eichlseder. Revisiting differential-linear attacks via a boomerang perspective with application to aes, ascon, clefia, skinny, present, knot, twine, warp, lblock, simeck, and SERPENT. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024*, LNCS, pages 38–72. Springer, 2024. doi:10.1007/978-3-031-68385-5_2.

[HNW26] Kai Hu, Zhongfeng Niu, and Meiqin Wang. Round-based approximation of (higher-order) differential-linear correlation - A geometric approach perspective. In Joan Daemen and Emmanuel Thomé, editors, *EUROCRYPT 2026*, LNCS, pages 152–180. Springer, 2026. doi:10.1007/978-3-032-25333-0_6.

[HZC+25] Kai Hu, Chi Zhang, Chengcheng Chang, Jiashu Zhang, Meiqin Wang, and Thomas Peyrin. Unlocking mix-basis potential: Geometric approach for combined attacks. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025*, LNCS, pages 293–334. Springer, 2025. doi:10.1007/978-3-032-01901-1_10.

[Jul14] JuliaMath contributors. Hadamard.jl: Fast multidimensional Walsh–Hadamard transforms for Julia. https://github.com/JuliaMath/Hadamard.jl, 2014. Accessed 2026-06-08.

[KLT15] Stefan Kölbl, Gregor Leander, and Tyge Tiessen. Observations on the SIMON block cipher family. Cryptology ePrint Archive, Paper 2015/145, 2015. URL: https://eprint.iacr.org/2015/145.

[KY95] Burton S. Kaliski and Yiqun Lisa Yin. On differential and linear cryptanalysis of the RC5 encryption algorithm. In Don Coppersmith, editor, *CRYPTO '95*, volume 963 of *LNCS*, pages 171–184. Springer, 1995. doi:10.1007/3-540-44750-4_14.

[LH94] Susan K. Langford and Martin E. Hellman. Differential-linear cryptanalysis. In *CRYPTO 1994*, volume 839 of *LNCS*, pages 17–25. Springer, 1994. doi:10.1007/3-540-48658-5_3.

[Mat93] Mitsuru Matsui. Linear cryptanalysis method for DES cipher. In Tor Helleseth, editor, *EUROCRYPT 1993*, volume 765 of *LNCS*, pages 386–397. Springer, 1993. doi:10.1007/3-540-48285-7_33.

[MCL+18] Stefano Markidis, Steven Wei Der Chien, Erwin Laure, Ivy Bo Peng, and Jeffrey S. Vetter. NVIDIA tensor core programmability, performance & precision. In *IPDPSW 2018*, pages 522–531. IEEE, 2018. doi:10.1109/IPDPSW.2018.00091.

[Met24] Meta PyTorch contributors. HadaCore: Tensor Core accelerated Hadamard transform kernels. https://github.com/meta-pytorch/applied-ai, 2024. Accessed 2026-08-31.

[MP09] Andreas Maurer and Massimiliano Pontil. Empirical Bernstein bounds and sample-variance penalization. In *COLT 2009 - The 22nd Conference on Learning Theory*, 2009. URL: https://arxiv.org/abs/0907.3740.

[NBGS08] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. Scalable parallel programming with CUDA. *ACM Queue*, 6(2):40–53, 2008. doi:10.1145/1365490.1365500.

[O'D14] Ryan O'Donnell. *Analysis of Boolean Functions*. Cambridge University Press, 2014. doi:10.1017/CBO9781139814782.

[Ope21] OpenMP Architecture Review Board. OpenMP application programming interface, version 5.2. https://www.openmp.org/specifications/, 2021. Accessed 2026-04-16.

[PAB+26] Léo Perrin, Jens Alich, Jules Baudrin, Aurélien Boeuf, Xavier Bonnetain, Alain Couvreur, Merlin Fruchon, Mathias Joly, Pierre Galissant, and Lukas Stennes. sboxU: black-box analysis of discrete functions. Available online at https://github.com/lpp-crypto/sboxU/, 2026.

[Riv94] Ronald L. Rivest. The RC5 encryption algorithm. In Bart Preneel, editor, *FSE 1994*, volume 1008 of *LNCS*, pages 86–96. Springer, 1994. doi:10.1007/3-540-60590-8_7.

[Sel00] Ali Aydin Selçuk. On bias estimation in linear cryptanalysis. In Bimal K. Roy and Eiji Okamoto, editors, *INDOCRYPT 2000*, volume 1977 of *LNCS*, pages 52–66. Springer, 2000. doi:10.1007/3-540-44495-5_6.

[Sha69] J. L. Shanks. Computation of the fast Walsh–Fourier transform. *IEEE Transactions on Computers*, C-18(5):457–459, 1969. doi:10.1109/T-C.1969.222685.

[SHS+17] Danping Shi, Lei Hu, Siwei Sun, Ling Song, Kexin Qiao, and Xiaoshuang Ma. Improved linear (hull) cryptanalysis of round-reduced versions of SIMON. *Sci. China Inf. Sci.*, 60(3):39101:1–39101:3, 2017. URL: https://doi.org/10.1007/s11432-015-0007-1, doi:10.1007/S11432-015-0007-1.

[SHSS16] Danping Shi, Lei Hu, Siwei Sun, and Ling Song. Linear(hull) cryptanalysis of round-reduced versions of KATAN. In *Proceedings of the 2nd International Conference on Information Systems Security and Privacy, ICISSP 2016, Rome, Italy, February 19–21, 2016*, pages 364–371. SciTePress, 2016. doi:10.5220/0005739103640371.

[TLH25] Cihangir Tezcan, Gregor Leander, and Hosein Hadipour. Cryptanalysis: Theory versus practice correcting cryptanalysis results on ascon, chacha, and serpent using gpus. *IACR Trans. Symmetric Cryptol.*, 2025(3):729–754, 2025. doi:10.46586/TOSC.V2025.I3.729-754.

[ZLZ23] Feng Zhang, Feng Li, and Wenzheng Zhang. Differential-linear cryptanalysis on simeck32/64 and simon32/64. *Journal of Physics: Conference Series*, 2504(1):012068, 2023. doi:10.1088/1742-6596/2504/1/012068.

[ZWH24] Yanyan Zhou, Senpeng Wang, and Bin Hu. MILP/MIQCP-based fully automatic method of searching for differential-linear distinguishers for SIMON-like ciphers. *IET Information Security*, 2024(1), 2024. doi:10.1049/2024/8315115.

A `FFTW` Route for Walsh-Hadamard Transforms

This appendix explains why a naive flat FFT does not compute the Walsh-Hadamard transform. It derives the valid indirect route through `FFTW`'s multidimensional Hartley transform and gives detailed timing comparisons.

Why a flat 1D FFT is not a WHT. The fast Fourier transform of length 2^n computes the characters of the cyclic group $\mathbb{Z}_{2^n}$. These characters are complex roots of unity $e^{2\pi ikj/2^n}$. The Walsh-Hadamard transform computes the characters of the elementary abelian group $\mathbb{F}_2^n$. These characters are the ± 1 functions $(-1)^{\alpha \cdot x}$ for $\alpha, x \in \mathbb{F}_2^n$. For $n \geq 2$, $\mathbb{Z}_{2^n}$ and $\mathbb{F}_2^n$ are non-isomorphic groups. Their characters are therefore structurally different. Thus, a flat 1D FFT gives incorrect WHT results.

The valid `FFTW` route. FFTW's real discrete Hartley transform of size two has the matrix

$$\mathrm{DHT}_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = H_1.$$

`FFTW` supports multidimensional transforms. We view the length-2^n array as an n-dimensional tensor with every dimension equal to two. Applying DHT_2 along each axis produces $H_1^{\otimes n} = H_n$, exactly the Walsh-Hadamard transform. This route reshapes the input, plans a multidimensional Hartley transform, and pays `FFTW`'s general-purpose planning overhead. It nevertheless produces mathematically correct results.

Example 1 ($n = 3$)**.** We interpret the flat array $[a_0, a_1, \ldots, a_7]$ as a $2 \times 2 \times 2$ tensor $A[b_2][b_1][b_0]$ with

$$A[b_2][b_1][b_0] = a_{4b_2+2b_1+b_0}.$$

The three size-two Hartley transforms along axes b_0, b_1, and b_2 correspond exactly to the three passes of the butterfly network shown in Figure 1.

Timing comparison. Table 21 reports Apple M4 timings for `LibFWHT` and the `FFTW` indirect route. Both use double-precision floating-point arrays. We exclude `FFTW`'s plan time and reuse a warm plan for every repetition. The comparison uses FFTW 3.3.10_2 on the Apple M4 with at most 10 threads. It plans with `FFTW_MEASURE`, reuses a warm plan, and excludes planning time, which is reported separately in the recorded data. Both libraries receive buffers from the same allocator, so their alignment is identical. We ran the full sweep twice; the single-thread results agree to within 0.04× at every size, and the table reports the second run. The versioned repository records the exact build and execution commands.

Table 21 and Table 22 give the complete sweeps. Single-thread `LibFWHT` is 2.5 to 3.3 times faster at all 23 sizes. With 10 threads it leads at 18 of the 23 sizes, and at $n = 30$ the two are level at 1.03×. We verified `LibFWHT` outputs against `FFTW` at every measured size.

B Extended Library Comparison Data

`LibFWHT` versus `sboxU`: Boolean Walsh-spectrum workloads

Table 23 gives Boolean Walsh-spectrum timings measured on one CPU core. The 2026-07-17 macOS run invokes sboxU's `cpp_walsh_transform` at upstream revision `7066b825c979`. It forces `LibFWHT` to the CPU backend and disables OpenMP on both paths. Before timing, the benchmark prepares equivalent inputs and checks exact coefficient agreement. The comparison is directly relevant to linear cryptanalysis and S-box analysis, which repeatedly

Table 21: Mean single-thread timings for LibFWHT and FFTW's multidimensional DHT route. Speedup is FFTW time divided by LibFWHT time.

Size	LibFWHT (ms)	FFTW (ms)	FFTW plan (ms)	Speedup (×)
2^{8}	0.0004	0.0014	0.062	3.26
2^{9}	0.0009	0.0027	0.069	3.07
2^{10}	0.0018	0.0053	0.076	3.01
2^{11}	0.0033	0.0101	0.079	3.08
2^{12}	0.0066	0.0213	0.103	3.23
2^{13}	0.0140	0.0449	0.143	3.19
2^{14}	0.0294	0.0934	0.161	3.18
2^{15}	0.0635	0.1940	0.170	3.06
2^{16}	0.1343	0.4038	0.187	3.01
2^{17}	0.2811	0.8289	0.198	2.95
2^{18}	0.5956	1.7265	0.212	2.90
2^{19}	1.2545	3.6061	0.250	2.87
2^{20}	2.6466	7.4727	0.259	2.82
2^{21}	5.5155	15.5974	0.316	2.83
2^{22}	11.4586	31.6354	0.325	2.76
2^{23}	24.3730	66.6453	0.329	2.73
2^{24}	51.2770	136.6046	0.366	2.66
2^{25}	108.6160	284.2278	0.405	2.62
2^{26}	231.8763	588.7123	0.416	2.54
2^{27}	480.8973	1218.8340	0.473	2.53
2^{28}	1009.4420	2532.1240	0.503	2.51
2^{29}	2112.6410	5228.0590	0.952	2.47
2^{30}	4419.4880	12 724.2690	0.679	2.88

compute Walsh spectra of large Boolean functions. For each size, the harness gathers 20 timing samples in an alternating order and calibrates each sample to at least 250 ms.

For Boolean truth tables with 2^{29} entries, LibFWHT takes 2.824 s per transform, compared with 5.675 s for sboxU, and is 2.00× faster. sboxU has no GPU backend, so CUDA timings are excluded from this comparison.

LibFWHT versus sboxU: batched Boolean workloads

The single-core comparison above isolates one transform at a time. A cryptanalytic search instead computes many spectra of the same size, so the batched case is the one that governs its running time. Table 24 reports that workload on a dual-socket AMD EPYC 9554 host, with both libraries given the same batch of 8 independent Boolean transforms across 8 workers. At 2^{10} through 2^{12} neither library threads, and both run a single transform at a time.

Across the full sweep from 2^{10} to 2^{27}, LibFWHT is faster at 14 of the 18 sizes. sboxU leads at 2^{16} through 2^{18}, by at most 1.20×. From 2^{19} upward LibFWHT leads at every size, by 1.86× to 3.70×, which is the regime a full-codebook search occupies.

pyfwht versus Hadamard.jl

Table 25 compares the Python package against the Julia implementation on the same double-precision inputs, with Hadamard.jl rescaled to the unnormalized convention and given 10 threads.

pyfwht leads at 19 of the 25 measured sizes. The large margins below 2^{20} reflect interpreter and call overhead in the Julia path rather than transform speed. From 2^{25}

Table 22: Ten-thread timings for `LibFWHT` and `FFTW`. Below 2^{20}, thread startup prevents a reproducible ranking. Columns follow Table 21.

Size	`LibFWHT` (ms)	`FFTW` (ms)	`FFTW` plan (ms)	Speedup (×)
2^{8}	0.0557	0.1282	0.806	2.30
2^{9}	0.0537	0.1328	0.580	2.47
2^{10}	0.1207	0.1309	0.726	1.08
2^{11}	0.1896	0.1324	0.852	0.70
2^{12}	0.1338	0.1295	1.033	0.97
2^{13}	0.1500	0.1314	2.997	0.88
2^{14}	0.1467	0.1393	1.995	0.95
2^{15}	0.1526	0.1688	2.163	1.11
2^{16}	0.2480	0.2020	2.374	0.81
2^{17}	0.2115	0.2914	2.552	1.38
2^{18}	0.3240	0.4526	2.763	1.40
2^{19}	0.4632	0.7932	3.047	1.71
2^{20}	0.7804	1.5412	3.319	1.97
2^{21}	1.4715	3.0845	3.598	2.10
2^{22}	4.0547	6.3033	3.814	1.55
2^{23}	11.2609	13.8439	4.196	1.23
2^{24}	24.9914	29.5702	4.504	1.18
2^{25}	54.4236	61.7812	4.936	1.14
2^{26}	121.6317	126.6310	5.170	1.04
2^{27}	258.3723	273.1517	5.601	1.06
2^{28}	565.1270	571.6710	5.962	1.01
2^{29}	1186.8440	1209.7830	6.629	1.02
2^{30}	2530.5020	2617.5510	6.976	1.03

Table 23: Median single-core timings for `LibFWHT` and `sboxU` on macOS. Speedup is the median paired ratio of `sboxU` time to `LibFWHT` time.

Size	`sboxU` (μs)	`LibFWHT` CPU (μs)	Speedup (×)
2^{10}	1.593	1.441	1.11
2^{11}	3.398	3.052	1.10
2^{12}	7.001	6.403	1.09
2^{13}	14.949	13.463	1.11
2^{14}	31.404	27.608	1.14
2^{15}	82.547	57.478	1.44
2^{16}	172.668	121.862	1.42
2^{17}	430.253	321.339	1.28
2^{18}	1115.480	859.252	1.30
2^{19}	2600.050	2049.480	1.27
2^{20}	5617.310	4432.510	1.27
2^{21}	11 670.000	9173.390	1.27
2^{22}	25 421.800	18 729.700	1.36
2^{23}	56 099.900	38 232.800	1.47
2^{24}	119 302.000	77 893.100	1.53
2^{25}	251 038.000	158 661.000	1.58
2^{26}	517 545.000	323 344.000	1.60
2^{27}	1 062 290.000	655 792.000	1.61
2^{28}	2 148 280.000	1 334 470.000	1.61
2^{29}	5 675 190.000	2 823 740.000	2.00

Table 24: Median per-transform timings for batched Boolean workloads. Speedup is the median ratio of `sboxU` time to `LibFWHT` time; outputs agree exactly.

Size	Workers	sboxU (μs)	LibFWHT (μs)	Speedup (×)
2^{10}	1	3.290	3.402	0.97
2^{11}	1	7.113	7.047	1.01
2^{12}	1	15.388	14.125	1.09
2^{13}	8	4.316	3.781	1.14
2^{14}	8	9.072	8.006	1.13
2^{15}	8	18.973	14.880	1.28
2^{16}	8	40.493	45.568	0.89
2^{17}	8	88.444	107.081	0.83
2^{18}	8	190.446	219.397	0.87
2^{19}	8	503.982	438.839	1.15
2^{20}	8	3471.953	937.364	3.70
2^{21}	8	10 949.370	4592.866	2.38
2^{22}	8	26 890.907	12 700.020	2.12
2^{23}	8	58 340.348	30 788.718	1.89
2^{24}	8	121 651.436	65 560.536	1.86
2^{25}	8	252 777.660	135 840.735	1.86
2^{26}	8	558 367.448	281 248.925	1.99
2^{27}	8	1 220 504.387	588 655.165	2.07

Table 25: Mean timings for `pyfwht` and `Hadamard.jl` over 5 repetitions. Speedup is `Hadamard.jl` time divided by `pyfwht` time.

Size	pyfwht (ms)	Hadamard.jl (ms)	Speedup (×)
2^{6}	0.0494	2.8066	56.82
2^{7}	0.0137	2.6445	193.38
2^{8}	0.0091	2.9539	323.72
2^{9}	0.0101	3.0607	302.04
2^{10}	0.0114	3.0129	265.07
2^{11}	0.0129	3.2005	248.58
2^{12}	0.0175	3.9009	223.12
2^{13}	0.0255	3.5321	138.65
2^{14}	0.0419	3.9397	94.01
2^{15}	0.0766	5.6109	73.26
2^{16}	0.1603	4.2866	26.75
2^{17}	0.3104	4.4619	14.38
2^{18}	0.4999	5.0216	10.05
2^{19}	0.5848	5.3792	9.20
2^{20}	0.9398	6.4519	6.87
2^{21}	1.9421	8.1773	4.21
2^{22}	5.6868	11.9826	2.11
2^{23}	13.5906	23.2794	1.71
2^{24}	29.0996	35.5113	1.22
2^{25}	66.3572	65.3574	0.98
2^{26}	137.3732	136.1156	0.99
2^{27}	315.0226	293.8673	0.93
2^{28}	681.7482	631.6317	0.93
2^{29}	1560.5170	1395.8897	0.89
2^{30}	6596.3711	4100.9530	0.62

upward `Hadamard.jl` leads once both implementations become bandwidth bound, by 1.02× at that size and 1.61× at 2^{30}.

`LibFWHT` versus `FFHT` on x86

`FFHT` is an x86-only library built around hand-written vector assembly for a single data type and one thread. Table 26 reports both the mismatched pairing used by its own benchmark, `LibFWHT int32_t` against `FFHT float`, and the datatype-matched pairing in double precision.

Table 26: Mean single-core timings on an AMD EPYC 7742. Ratios divide `LibFWHT` time by `FFHT` time.

	`int32_t` versus `float`			`double` versus `double`		
Size	`LibFWHT` (μs)	`FFHT` (μs)	×	`LibFWHT` (μs)	`FFHT` (μs)	×
2^{8}	1.708	0.207	8.27	0.803	0.199	4.04
2^{9}	3.337	0.159	21.00	1.844	0.302	6.11
2^{10}	3.632	0.501	7.25	3.520	0.660	5.34
2^{11}	6.564	0.882	7.44	7.534	1.462	5.15
2^{12}	13.177	1.717	7.68	15.599	3.306	4.72
2^{13}	26.832	4.268	6.29	32.818	7.073	4.64
2^{14}	55.959	8.273	6.76	58.604	17.182	3.41
2^{15}	124.247	17.142	7.25	143.845	31.313	4.59
2^{16}	234.476	38.600	6.08	298.655	65.129	4.59
2^{17}	482.195	78.669	6.13	629.798	134.880	4.67
2^{18}	998.855	157.649	6.34	1295.249	344.423	3.76
2^{19}	2087.921	393.470	5.31	2705.451	711.534	3.80
2^{20}	4249.548	805.822	5.27	5595.390	1476.348	3.79
2^{21}	8871.916	1652.974	5.37	11 931.778	3156.550	3.78
2^{22}	22 655.214	3592.412	6.31	24 851.788	7822.541	3.18
2^{23}	59 237.944	7684.323	7.71	53 092.456	15 830.160	3.35
2^{24}	120 808.036	16 642.000	7.26	112 661.775	36 161.040	3.12
2^{25}	247 890.307	35 302.618	7.02	238 109.793	75 186.950	3.17

With the data types matched, `FFHT` leads by 3.12× to 6.11× across the sweep, against 5.27× to 21.00× in the mismatched pairing. This is the expected outcome of a narrow design point: `FFHT` supports one data type, one instruction-set family, and one thread, whereas the same `LibFWHT` source serves integer, floating-point, and Boolean inputs on processors and graphics processors alike.

`pyfwht` versus a reference Python implementation

Table 27 compares `pyfwht` against SymPy's reference transform. The comparison measures the cost of a pure-Python reference rather than a competing implementation, and is included because that reference is a common starting point in Python workflows.

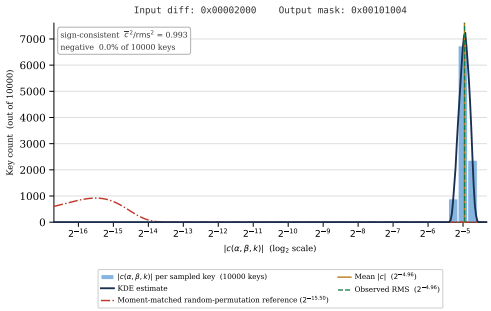
C Per-Key Correlation Distributions

This appendix collects the per-key correlation distributions of every distinguisher reported in Section 7, in the order of the cipher targets there. Each panel plots the absolute correlation of one fixed pair over independently sampled master keys, on the logarithmic axis described in Subsection 6.3. The two distributions that the key-dependence analysis

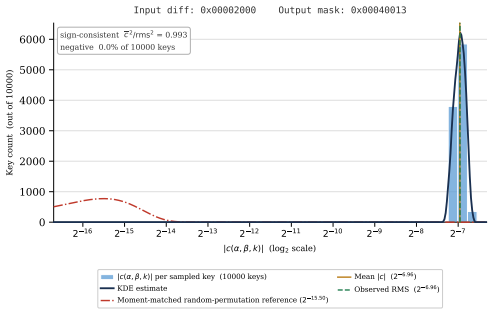
Table 27: Mean timings for `pyfwht` and SymPy, with exact output agreement at every size.

Size	**SymPy (ms)**	**pyfwht (ms)**	**Speedup (×)**
2^{6}	0.4726	0.0008	579
2^{7}	1.0607	0.0008	1 254
2^{8}	2.4253	0.0013	1 866
2^{9}	5.2977	0.0015	3 561
2^{10}	11.5804	0.0027	4 349
2^{11}	25.6905	0.0048	5 361
2^{12}	53.9169	0.0085	6 349
2^{13}	116.3457	0.0156	7 470
2^{14}	249.7740	0.0360	6 935
2^{15}	532.3777	0.0745	7 149
2^{16}	1152.2247	0.1531	7 528
2^{17}	2452.1273	0.3121	7 857
2^{18}	5330.2495	0.6756	7 889
2^{19}	11 465.6307	1.4043	8 165

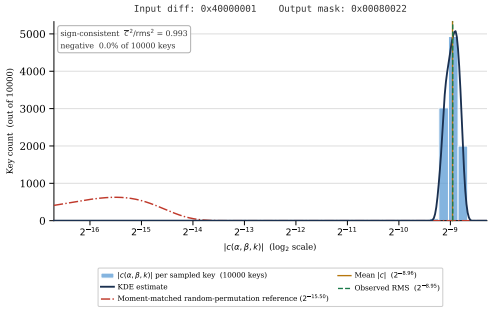
works on directly, Figure 7 and Figure 9, stay in Section 8 beside the text that analyzes them.



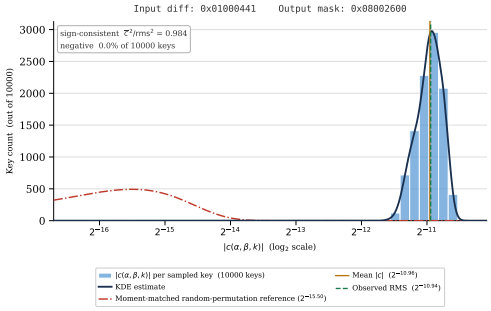
(a) 10-round DL



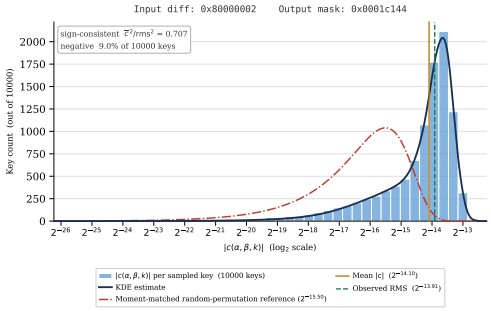
(b) 11-round DL



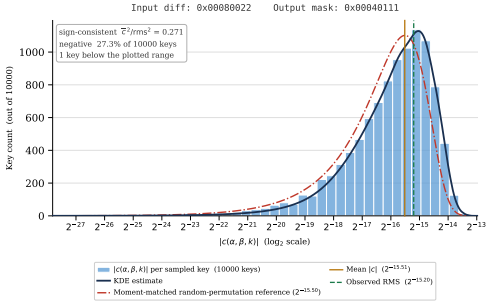
(c) 12-round DL



(d) 13-round DL

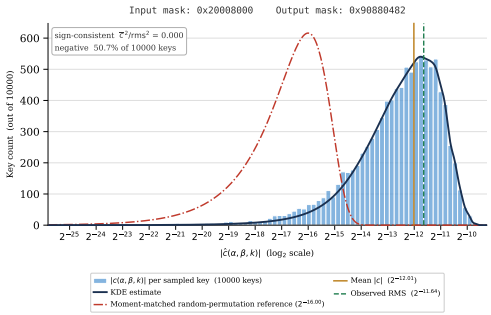


(e) 14-round DL

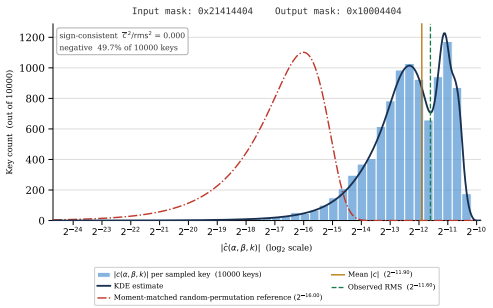


(f) 15-round DL

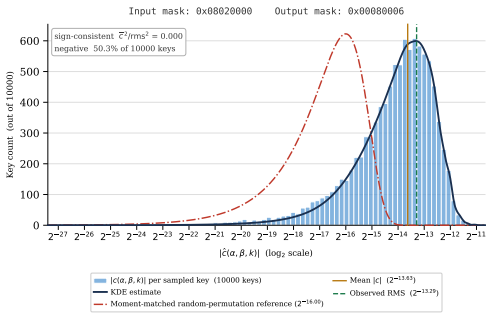
Figure 10: Per-key absolute DL correlation distributions for six selected reduced-round SIMON-32 distinguishers. Each panel states its input difference and output mask. Every panel uses $K_{\mathrm{conf}} = 10000$ independently sampled master keys. The 15-round panel shows the published pair [CXZQ25] recovered by the fixed-input sweep.



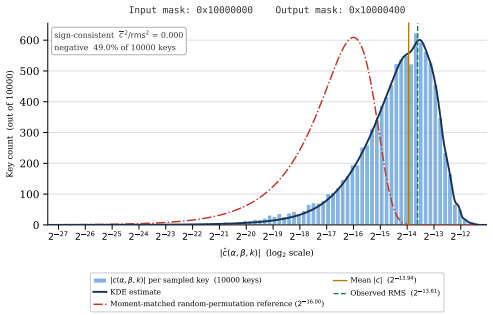
(a) 9-round linear



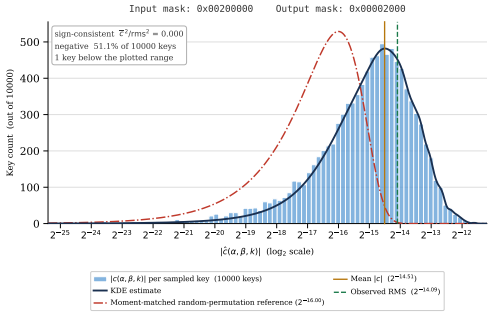
(b) 10-round linear



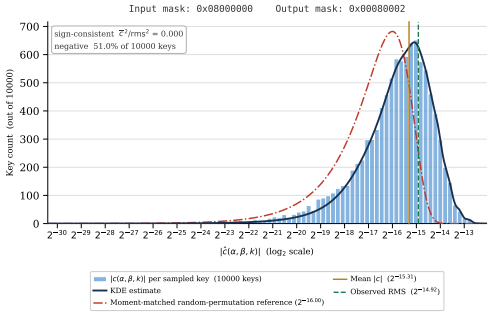
(c) 11-round linear



(d) 12-round linear

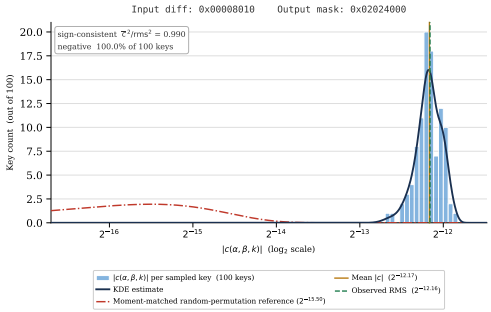


(e) 13-round linear

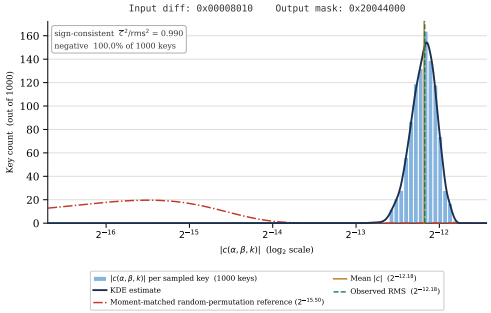


(f) 14-round linear

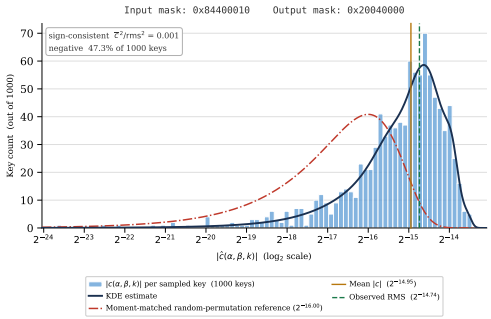
Figure 11: Per-key absolute linear-correlation distributions for six selected reduced-round SIMON-32 pairs. Each panel states its input and output masks. Each panel uses $K_{\mathrm{conf}} = 10000$ independently sampled master keys.



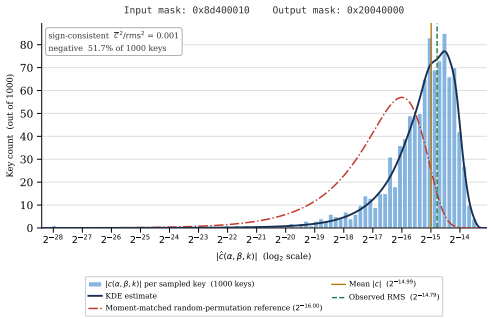
(a) 80-round selected DL distinguisher



(b) 84-round selected DL distinguisher

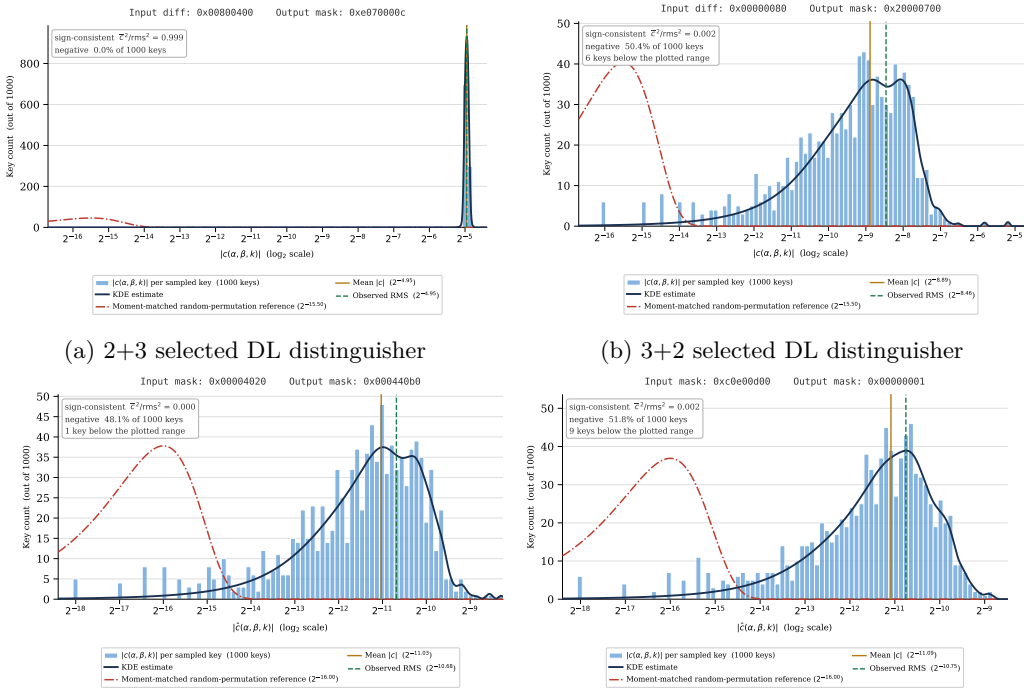


(c) 84-round published linear distinguisher



(d) 84-round selected linear distinguisher

Figure 12: Per-key absolute-correlation distributions for selected KATAN-32 distinguishers with start round zero. The top panels show the selected local DL distinguishers, while the bottom panels show the published 84-round linear distinguisher and the selected local linear distinguisher from Table 6. The 80-round DL panel uses 100 independently sampled master keys; every other panel uses 1,000.



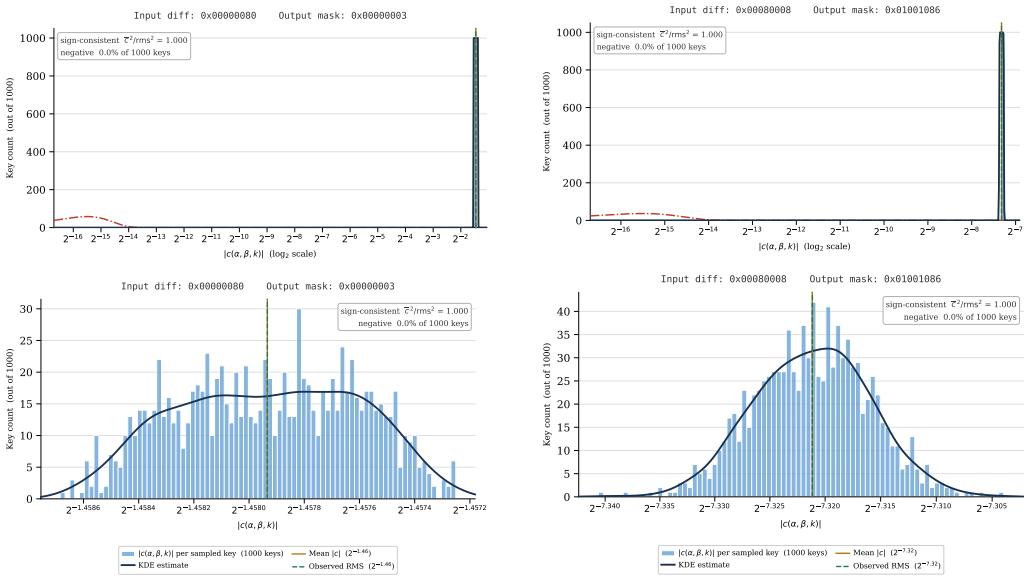
(a) 2+3 selected DL distinguisher

(b) 3+2 selected DL distinguisher

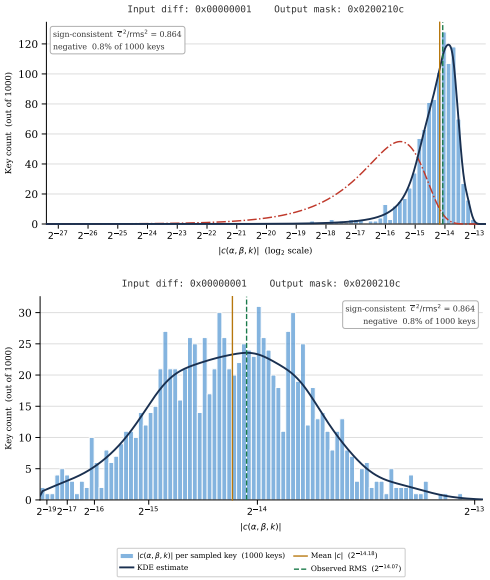
(c) 2+3 selected linear distinguisher

(d) 3+2 selected linear distinguisher

Figure 13: Per-tuple absolute-correlation distributions for the reported spectral U-shape BEANIE results below 3+3 rounds from Table 7. Both branches are suffixes of the five-round parent, including final whitening. Each panel uses 1,000 independently sampled (K, T_1, T_2) tuples and directly evaluates its complete 2^{32}-input codebook.

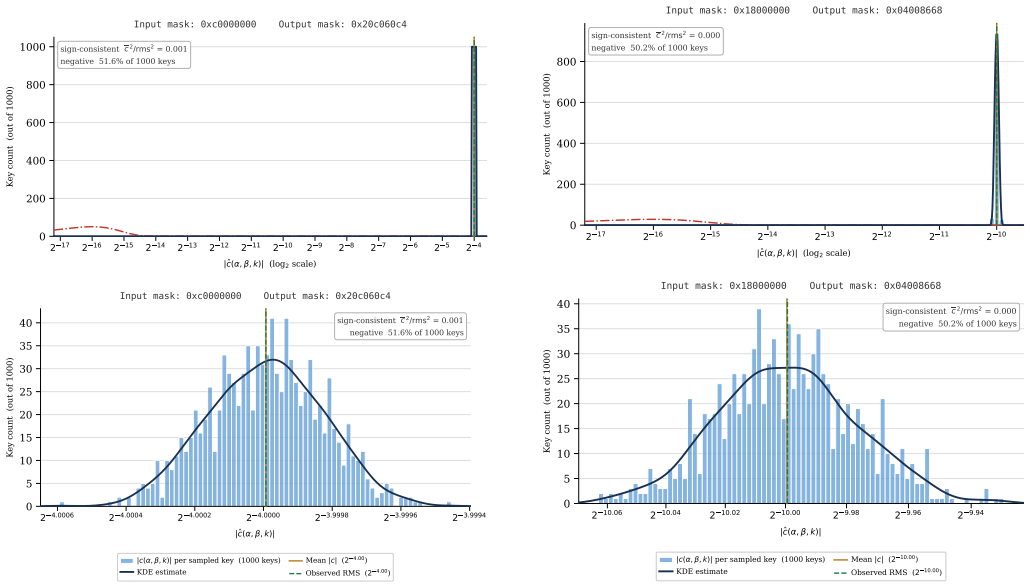


(a) 64-cycle DL (b) 100-cycle DL



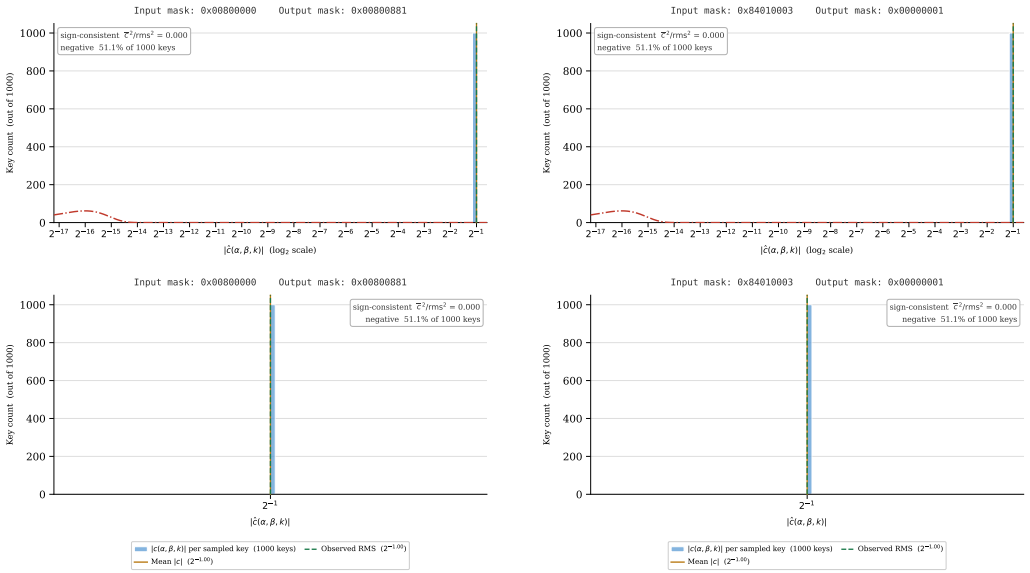
(c) 128-cycle DL

Figure 14: Per-key absolute-correlation distributions for the selected KeeLoq DL pairs of Table 8. Each panel shows the same 1,000 keys twice: above on a logarithmic magnitude axis, against the random-permutation reference of Subsection 6.2, and below on a linear axis over the range the data occupies.



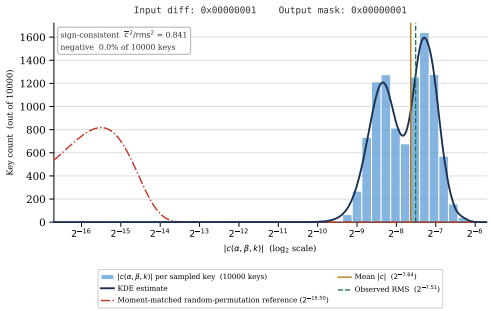
(a) 64-cycle linear (b) 100-cycle linear

Figure 15: Per-key absolute-correlation distributions for the selected KeeLoq linear pairs of Table 8, in the two views of Figure 14.

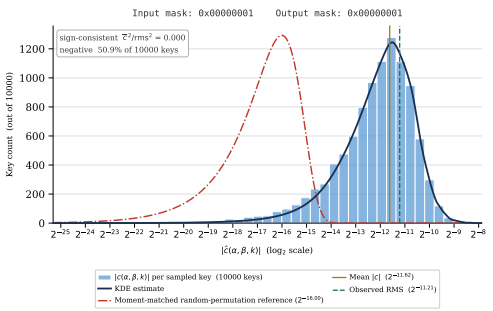


(a) 32-cycle, input mask fixed (b) 32-cycle, output mask fixed

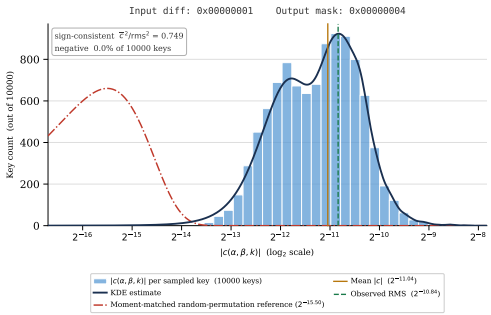
Figure 16: The two independently confirmed 32-cycle KeeLoq representatives, in the two views of Figure 14. Every sampled key has $|c| = 2^{-1.00}$, so the mean and RMS coincide and a kernel-density estimate is undefined.



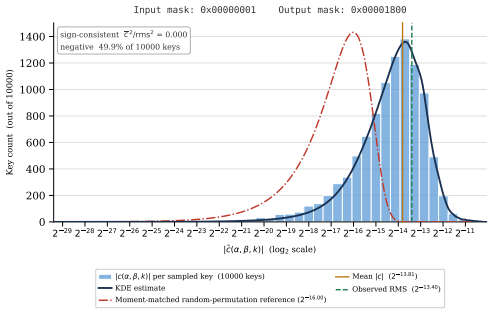
(a) 4-round DL



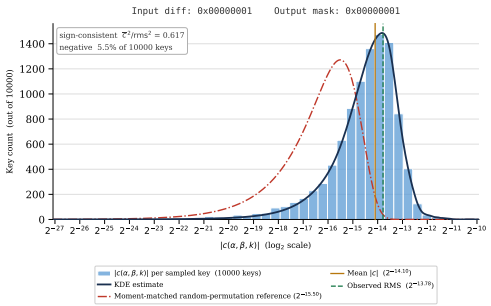
(b) 4-round linear



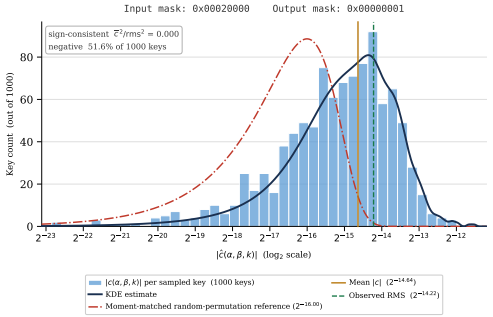
(c) 5-round DL



(d) 5-round linear



(e) 6-round DL



(f) 6-round linear

Figure 17: Per-key absolute-correlation distributions for the RC5-16 distinguishers in Table 9. Each row shows one round count, with DL on the left and linear analysis on the right. The six-round linear panel uses 1,000 independently sampled master keys; the others use 10,000. The linear signs are nearly balanced, whereas the DL signed mean retains $\rho = 0.841$, 0.749, and 0.617 of the second moment at four, five, and six rounds, respectively. At six rounds, 5.51% of the DL correlations are negative, so sign stability is not universal.

D A Partial Schedule-Aware Analysis Beyond 32 Bits

The exact experiments in the main text use 32-bit blocks because they evaluate the complete codebook. To test the geometric procedure beyond that regime, we considered the six-round middle of the published 16-round SIMON48/96 trail with configuration $(5, 6, 5)$ [CXZQ25]. Its boundary difference is $(\texttt{0x000022}, \texttt{0x000008})$, and its boundary mask is $(\texttt{0x000004}, \texttt{0x000000})$. The source gives the continuous-difference estimate $2^{-3.01}$ for this middle. This appendix analyzes only that middle, not the complete 16-round distinguisher. We instantiate the six-round map with $k_0, \ldots, k_5$ of the standard key schedule, as in the trail model below.

To assess its key dependence, we estimated the signed correlation for a fresh sample of 1,000 master keys using 2^{20} uniformly drawn input pairs per key, with replacement. The confidence analysis assumes independent uniform sampling of the keys and independent input samples for each key. All correlation estimates were positive, with RMS $2^{-1.9326}$, signed mean $2^{-1.9329}$, and a coefficient of variation of 2.05%. At this correlation magnitude, input sampling contributes a relative standard deviation of about 0.36%, substantially less than the observed spread. The measurements therefore indicate a narrow distribution with residual key dependence, without determining its exact shape.

The same measurements allow constant-coefficient dominance to be distinguished from key independence. Let $Y_1, \ldots, Y_M \in [-1, 1]$, with $M = 1000$, denote the per-key correlation estimates. Write $\bar{Y}$ and $\overline{Y^2}$ for their sample mean and mean square, and s^2_Y and $s^2_{Y^2}$ for the unbiased sample variances of the estimates and their squares. Unbiased input sampling implies $\mathbb{E}[Y \mid k] = f(k)$ and $\mathbb{E}Y^2 \geq \mathbb{E}_k f(k)^2$. Thus a lower bound on $\mathbb{E}Y$ and an upper bound on $\mathbb{E}Y^2$ suffice to bound ρ_{key}. With the positive-correlation direction fixed before this experiment, the empirical Bernstein inequality [MP09, Theorem 4], applied to $-Y$ and Y^2, yields one-sided bounds with failure probability 0.025 each. Writing $t = \ln 80$, these bounds are

$$L = \bar{Y} - \sqrt{\frac{2s^2_Y t}{M}} - \frac{14t}{3(M-1)}, \qquad U = \overline{Y^2} + \sqrt{\frac{2s^2_{Y^2} t}{M}} + \frac{7t}{3(M-1)}.$$

A union bound establishes $a_0 \geq L$ and $\mathbb{E}_k f(k)^2 \leq U$ simultaneously with 95% confidence. For this sample, $L > 0.2409$ and $U < 0.07912$, implying $\rho_{\text{key}} \geq L^2/U > 0.733$ on the same confidence event. It follows from Equation 7 that every nonconstant master-key coefficient has magnitude less than $0.604|a_0|$. This simultaneous bound accounts for uncertainty from both key and input sampling, but does not establish negligible total key dependence or exclude rare weak keys.

For a structural explanation, we also analyzed the mix-basis expansion after mapping its round-key characters to master-key characters through the affine SIMON48/96 key schedule. The basis switch was placed after round 3. Here the weight of a trail contribution c_t is $-\log_2 |c_t|$, as in Subsection 8.2. Exhaustive enumeration through weight 22 returned 82,944 trails. The 39,936 trails on the zero master-key mask summed to $+2^{-7.193}$. The remaining 43,008 trails occupied 11 nonzero master-key masks, and every one of those 11 grouped sums was exactly zero. At the minimum key-dependent weight 21, the 7,168 trails further split into 448 complete families of 16 trails. Within each family, four free switch-boundary bits covered all 2^4 settings and the trail sign was a nonconstant affine character of those bits. Each family therefore summed to zero by Walsh orthogonality, which certifies the cancellation at that order rather than merely observing it numerically.

A separate enumeration restricted to the zero master-key mask was exhaustive through weight 23. Its 65,536 trails summed to $+2^{-6.660}$, still about 4.7 bits below the sampled middle correlation. The partial sums have therefore not converged. This case combines exact leading-order cancellations with statistical constant-coefficient dominance beyond a 32-bit block. The exact correlation and control of the unenumerated key-dependent tail remain open.

E Cipher Specifications

This appendix summarizes the round functions, key schedules, and encoding conventions of the five 32-bit targets in Section 7. The cited specifications provide the remaining design details and constants. Bit zero is the least-significant bit, state and key additions are XORs unless stated otherwise, and & denotes bitwise AND.

E.1 SIMON-32

SIMON-32 is the smallest member of the SIMON block cipher family [BSS+15]. The block is a pair of 16-bit words (x, y), encoded as $2^{16}x + y$. The 64-bit master key supplies the initial round keys $k_0, \ldots, k_3$, from its least-significant to its most-significant 16-bit word. The full cipher runs 32 rounds.

With round key k_i, the Feistel map is

$$f(x) = (x \lll 1) \,\&\, (x \lll 8) + (x \lll 2),$$
$$(x, y) \leftarrow \big(y + f(x) + k_i,\ x\big),$$

where $+$ is componentwise addition in $\mathbb{F}_2^{16}$, & denotes bitwise AND, and every shift is a left circular rotation on a 16-bit word. Figure 18 shows one round.

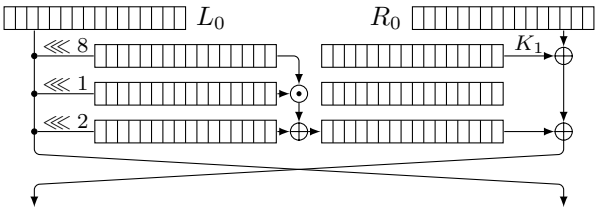


Figure 18: The SIMON-32 round function.

The key schedule uses the constant $c = 2^{16} - 4 =$ `0xfffc` and the first 28 bits of the specified sequence z_0, read from left to right:

$$(z_0)_0 \cdots (z_0)_{27} = \texttt{11111010001001010110000111}00.$$

For $0 \le i < 28$, the recurrence is

$$k_{i+4} \leftarrow c + (z_0)_i + k_i + (I + S^{-1})\big(S^{-3}k_{i+3} + k_{i+1}\big),$$

where I is the identity, S^{-j} denotes a right rotation by j bits on 16-bit words, and $(z_0)_i$ is added in the least-significant bit.

E.2 KATAN-32

KATAN-32 is the 32-bit member of the KATAN family [CDK09]. The state consists of two coupled nonlinear feedback shift registers L_1 and L_2, of lengths 13 and 19 bits, respectively. We encode L_2 in bits 0–18 and L_1 in bits 19–31, with each register indexed from its least-significant bit. The key length is 80 bits, and the full cipher runs 254 rounds. The master-key bits initialize $k_0, \ldots, k_{79}$; the remaining key bits satisfy

$$k_i = k_{i-80} + k_{i-61} + k_{i-50} + k_{i-13}, \qquad 80 \le i < 508.$$

Round r uses k_{2r}, k_{2r+1} and the public irregular rule bit IR_r generated by the specified 8-bit round counter [CDK09, Sections 3–4]. For $0 \le r < 254$, the feedback bits are

$$f_a = L_1[12] + L_1[7] + (L_1[8] \,\&\, L_1[5]) + (L_1[3] \,\&\, \mathrm{IR}_r) + k_{2r},$$
$$f_b = L_2[18] + L_2[7] + (L_2[12] \,\&\, L_2[10]) + (L_2[8] \,\&\, L_2[3]) + k_{2r+1},$$

Each register shifts toward higher indices, discarding its highest bit; f_b enters $L_1[0]$ and f_a enters $L_2[0]$. The rule bits are indexed by the absolute round, not restarted for an internal round window.

E.3 BEANIE

BEANIE is a tweakable substitution-permutation-network cipher with a 32-bit block, a 128-bit master key K, and a 128-bit tweak T [GHM+25, Section 3]. Its state has four rows and two columns of 4-bit cells, indexed down each column, with cell zero the most-significant nibble of the hexadecimal word. The standard variant has five datapath rounds and five tweakey-schedule rounds; the seven-round variant increases both counts to seven. Writing ART_i, SC, SR, and MC for addition of the 32-bit round tweakey k_i, cell substitution, ShiftRows, and MixColumns, respectively, the five-round datapath uses

$$R_i = \mathsf{MC} \circ \mathsf{SR} \circ \mathsf{SC} \circ \mathsf{ART}_i \quad (0 \le i < 4), \qquad R'_4 = \mathsf{ART}_5 \circ \mathsf{SR} \circ \mathsf{SC} \circ \mathsf{ART}_4.$$

Thus the final encryption round omits MixColumns, and its inverse omits inverse MixColumns. SubCells applies the S-box S, whose outputs in hexadecimal are

$$(S(0), \ldots, S(15)) = (0, 4, 2, \mathtt{b}, \mathtt{a}, \mathtt{c}, 9, 8, 5, \mathtt{f}, \mathtt{d}, 3, 7, 1, 6, \mathtt{e}).$$

ShiftRows swaps the two cells in rows 1 and 3, numbered from zero. MixColumns multiplies each column by

$$M = \begin{pmatrix} 2 & 1 & 1 & 9 \\ 1 & 4 & \mathtt{f} & 1 \\ \mathtt{d} & 9 & 4 & 1 \\ 1 & \mathtt{d} & 1 & 2 \end{pmatrix} \quad \text{over } \mathbb{F}_2[t]/(t^4 + t + 1),$$

with hexadecimal field elements.

The nonlinear tweakey schedule starts from T. Each of its five rounds adds K and a round constant, applies the S-box and PRINCE-based linear layers to two 64-bit halves, and mixes those halves. A final addition of K and the last round constant gives four 32-bit words (k_0, k_1, k_2, k_3), in most-significant-word-first order; the remaining keys are $k_4 = k_0 + k_1$ and $k_5 = k_2 + k_3$ [GHM+25, Section 3.2]. For the U-shape windows of Table 7, both branches retain this full schedule under the same K and distinct tweaks. For example, 2+3 uses k_3, k_4, k_5 in encryption and k'_5, k'_4, k'_3, k'_2 in decryption, in application order, where primes denote the second tweak. These are suffixes joined at the hidden ciphertext, as in the designers' reduced-round distinguisher convention [GHM+25, Section 5].

E.4 KeeLoq

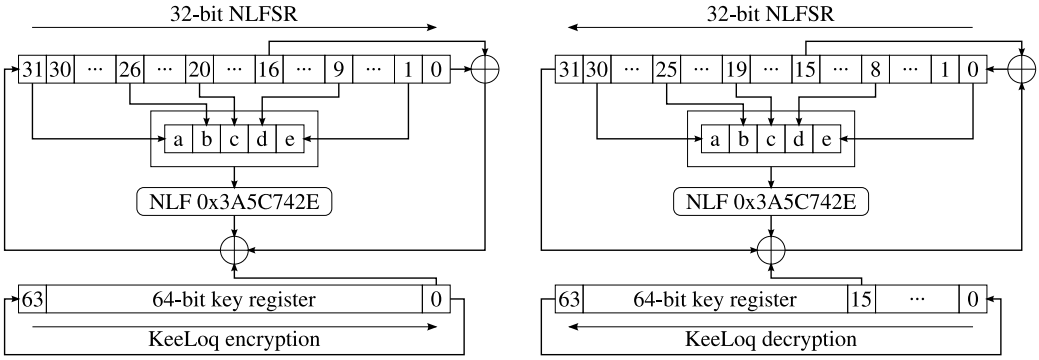
KeeLoq uses a 32-bit state $X = (x_{31}, \ldots, x_0)$, a 64-bit master key $K = (k_{63}, \ldots, k_0)$, and 528 encryption cycles [Bog07]. At cycle i, it computes

$$u_i = k_{i \bmod 64} \oplus x_{16} \oplus x_0 \oplus \mathrm{NLF}(x_{31}, x_{26}, x_{20}, x_9, x_1),$$

then updates the state as

$$(x_{31}, x_{30}, \ldots, x_0) \leftarrow (u_i, x_{31}, x_{30}, \ldots, x_1).$$

Figure 19 shows one encryption cycle and one decryption cycle. The five-input nonlinear feedback function is represented by the truth-table constant `0x3a5c742e`: $\mathrm{NLF}(a, b, c, d, e)$ is bit $16a + 8b + 4c + 2d + e$ of that constant. Decryption reverses the shift and applies the key bits in reverse cycle order.



(a) Encryption cycle (b) Decryption cycle

Figure 19: KeeLoq encryption and decryption cycles from the upstream reference implementation [had26].

E.5 RC5-16

RC5-$w/r/b$ encrypts two w-bit words with r rounds and a b-byte key [Riv94]. We use $w = 16$ and $b = 16$, encoding the state (A, B) as $A + 2^{16}B$. Here $+$ denotes addition modulo 2^{16} within each word, and $\oplus$ denotes XOR. We write (A_0, B_0) for the plaintext and (A_i, B_i) for the state after complete round i. The key schedule expands the master key into $t = 2(r + 1)$ 16-bit subkeys $S_0, \ldots, S_{t-1}$. After whitening $A \leftarrow A + S_0$, $B \leftarrow B + S_1$, round $i = 1, \ldots, r$ performs the sequential updates

$$A \leftarrow ((A \oplus B) \lll B) + S_{2i}, \qquad B \leftarrow ((B \oplus A) \lll A) + S_{2i+1},$$

where rotation amounts are reduced modulo 16 and the second update uses the new A. Decryption reverses these updates using subtraction and right rotation.

The standard key schedule packs the 16 key bytes into eight little-endian words $L_0, \ldots, L_7$. It initializes the subkey words by $S_j = \texttt{0xb7e1} + j \cdot \texttt{0x9e37}$ for $0 \le j < t$ and mixes the two arrays for $3\max(t, 8)$ iterations using the specified additions and rotations [Riv94, Section 4.3]. Our r-round instances include whitening and all r complete rounds, with no additional half-round. Each instance expands its own $2(r + 1)$ subkeys rather than taking a prefix of a longer instance's expanded key.