

An Input-Fed AES-Round Family for Fixed-Length Hashing

Peter Rindal

August 2026

Abstract

Secure computation often hashes high-entropy fixed-length values in inner loops. A public permutation with terminal Davies–Meyer feed-forward can expose the permutation value when an application applies another feed-forward step. We study an input-fed alternative that injects the input after every AES round.

The base construction, $H_{128} : \{0, 1\}^{128} \rightarrow \{0, 1\}^{128}$, uses ten full AESENC-style rounds with keys $x + c_i$ and no AES key expansion. Its concrete analysis adapts differential, collision, linear, integral, and meet-in-the-middle (MITM) methods to this shared-input round-key interface. We prove a three-round integral distinguisher and give a five-round anchored MITM attack candidate with estimated time 2^{120} ; the remaining results provide reduced-round evidence rather than a proof of random-function behavior. On an AMD Ryzen 9 7950X, measured scalar and eight-lane AES-NI costs are within one percent of fixed-key AES-128; a sixteen-block VAES kernel is 2.3% slower.

Two extensions use the same input-fed principle. The tweakable construction H_{128}^{tw} splits a 72-bit public counter into an eight-bit tweak and a 64-bit epoch. One public schedule serves 256 consecutive positions, and consecutive fixed-key AES outputs under a published key supply the epoch schedules. Within one epoch, a pair with different tweaks can cancel at most one complete boundary injection. The construction also admits an exact 256-query two-round integral distinguisher. The wide construction H_{256} maps 256 bits to 256 bits. It uses two AES lanes and adopts the column permutation and round constants of Haraka-256 v2 [69]. We analyze the differential, integral, collision, slide, rebound, and MITM mechanisms introduced by these extensions.

A supporting ideal-permutation analysis shows that affine wrappers with one or two permutation calls fail generically. For three calls with independently uniform public constants, it gives a birthday-order bound when the distinguisher completes its primitive queries before querying the target. Finally, we define checkable punctured vectors and prove a GGM-style protocol secure against static semi-honest corruptions in the ideal-OT and random-oracle model.¹

¹This work was created with the assistance of ChatGPT versions 5.6 and 6.0.

1 Introduction

Secure multiparty computation (MPC) protocols repeatedly hash short, high-entropy values. Typical inputs include 128-bit wire keys, OT correlations, and seeds in a GGM tree. These inputs need not be independent. An adversary may know affine relations among them even when each hidden offset has high entropy. Correlation-robust hashing was introduced for OT extension [74]. Related requirements arise in garbled circuits [18, 104], silent OT and vector oblivious linear evaluation (VOLE) [8], private set intersection [90], and distributed point functions [17].

A general-purpose hash such as SHA-3 provides a conservative random-oracle interface. Its variable-length interface and sponge machinery are often more than an inner MPC loop needs. Because dedicated AES instructions make fixed-key AES fast, implementations often hash one block by applying it as a public permutation followed by terminal feed-forward.

Terminal feed-forward does not give a random-function interface that composes safely with every application. Let P be a public permutation and define

$$H_{\text{DM}}(x) := P(x) + x.$$

Suppose a downstream construction expands x as

$$G_H(x) := (H(x), H(x) + x).$$

Substituting H_{DM} gives

$$G_{H_{\text{DM}}}(x) = (P(x) + x, P(x)).$$

The second component exposes the public permutation value. An observer with access to P^{-1} can recover x . This example does not contradict narrower correlation-robust guarantees for permutation feed-forward. It shows why such a map is not a drop-in random function for applications that apply another feed-forward step.

We instead inject the input throughout an AES-round computation. Each layer applies a full AES round and then injects the input with a public constant. This structure removes the immediate identity above while retaining a direct mapping to AES instructions. The resulting functions are public and unkeyed. Each construction aims to emulate a uniform random function on its stated fixed domain and range.

The family has three members:

$$\begin{aligned} H_{128} &: \{0, 1\}^{128} \longrightarrow \{0, 1\}^{128}, \\ H_{128}^{\text{tw}} &: \{0, 1\}^{128} \times [0, 256) \longrightarrow \{0, 1\}^{128}, \\ H_{256} &: \{0, 1\}^{256} \longrightarrow \{0, 1\}^{256}. \end{aligned}$$

The base construction, H_{128} , injects one 128-bit input through ten AES rounds. The tweakable construction, H_{128}^{tw} , additionally injects a coded eight-bit tweak

at every boundary. For a streaming public counter, the low byte is the tweak and the upper 64 bits select an epoch schedule. The wide construction, H_{256} , injects two input blocks into two AES lanes and applies the Haraka-256 v2 column permutation every two layers [69]. An application that needs only 128 output bits may truncate the wide output. Truncation is not a separate construction. The three members share a design method, not a security theorem.

The family raises two security questions. The concrete question asks whether repeated input injection, public tweaks, or the column permutation create attacks that do not arise in ordinary AES. The structural question replaces the AES round with a reused ideal permutation and asks whether the affine wrapper has an immediate generic failure. We address the first question through concrete cryptanalysis. A supporting ideal-permutation result addresses the second.

1.1 Contributions

Constructions and implementation. We specify the three functions, their constant schedules, and test vectors, and provide scalar AES-NI, eight-lane AES-NI, and sixteen-block VAES kernels for H_{128} . The base construction uses one initial XOR and ten AESENC instructions, without per-input key expansion. On an AMD Ryzen 9 7950X, its scalar and eight-lane costs are within one percent of fixed-key AES-128; VAES is 2.3% slower. Code, validation artifacts, and supplementary analyses accompany the paper [1].

Concrete cryptanalysis. We adapt differential, collision, linear, integral, and meet-in-the-middle techniques to the input-tied round keys. For H_{128} , we prove a three-round 2^{16} -query integral distinguisher and give a five-round anchored MITM attack candidate with estimated time 2^{120} . For H_{128}^{tw} , we prove that distinct tweaks can cancel at most one complete injection within an epoch, and give a two-round 256-query integral distinguisher. For H_{256} , we analyze cross-lane collision, rebound, slide, and MITM mechanisms. This analysis includes whether a published full-round Haraka-512 v2 partial-preimage attack transfers. Section 6 gives the quantitative results and their model boundaries.

Structural support. For one reused public ideal permutation, one and two affine calls admit generic attacks. Three calls with independently uniform public constants achieve a birthday-order bound when all primitive queries precede the target queries. This result concerns the affine wrapper rather than the concrete AES-round family.

Applications. Punctured GGM trees support silent OT and pseudorandom correlation generators [57, 8, 17]. We define checkable punctured vectors and prove a GGM-style protocol secure against static semi-honest corruptions in the ideal-OT and random-oracle model. The punctured party learns all but one leaf and can test guesses for the missing value. Where this interface suffices, an n -leaf tree costs approximately n calls. This is one third fewer than the

pseudorandom-leaf Half-Tree [62] and matches its correlation-specific variant. We also describe the direct semi-honest OT application and discuss other uses of short-input hashing.

1.2 Claims and Evidence

The concrete family is new and has not yet received broad public cryptanalysis. This paper provides reproducible evidence rather than a proof that the concrete functions behave as random functions. The analyses do not give full-round differential, linear, or integral bounds, and the anchored MITM model omits guessed intermediate values and superposition states. All concrete claims concern the specified constant schedules. Auxiliary-schedule calculations are identified separately as checks of sensitivity to constants. The claims do not quantify over arbitrary round constants. Behavior across many instance identifiers or epochs also remains open. Appendix E explains the resulting requirement on parameter generation.

Table 1 separates the target interfaces, proved results, and cryptanalytic evidence. Later sections give local limitations when they materially affect a result.

Object	Target and evidence
H_{128}	Its target is a uniform random function from 128 bits to 128 bits. The paper gives reduced-round cryptanalysis and reports no full-round attack. It does not prove the target behavior.
H_{128}^{tw}	For each fixed epoch, its target is a uniform random function on $\{0, 1\}^{128} \times [0, 256)$. Equivalently, the ideal target has 256 independent tweak slices. The paper gives exact structural results and reduced-round cryptanalysis, but no proof of the target behavior.
H_{256}	Its target is a uniform random function from 256 bits to 256 bits. The paper analyzes the new cross-lane structure and a published Haraka partial-preimage technique. It proves neither the target behavior nor full-round bounds for collisions, fixed-target preimages, or partial preimages.
Checkable punctured vectors	The protocol realizes its stated functionality against static semi-honest corruptions in the ideal-OT and random-oracle model. For a uniform missing value independent of the adversary’s initial view, q checks and one final guess succeed with probability at most $(q+1)/2^k$. Replacing the oracle by a concrete family member invokes the corresponding unproved target claim.
Affine ideal-permutation cascade	Depths one and two fail. With independently uniform public constants, depth three has a birthday-order bound when the distinguisher completes its primitive queries before querying the target.

Table 1: Scope of the principal claims and evidence.

The paper first specifies the family and reports the measured performance of H_{128} . It then evaluates each member against its target interface and generic baseline. The ideal-permutation analysis provides supporting evidence. Applications appear last, with detailed proofs in the appendices.

2 Preliminaries and Context

We fix notation and relate the family to earlier designs.

2.1 Notation and Interfaces

In the idealized analyses and applications, $\kappa \geq 1$ is the security parameter and the block or output length. The concrete base construction fixes $\kappa = 128$. The wide construction states its 256-bit domain and range explicitly. The assignment $x := y$ defines or updates x , whereas $x = y$ denotes equality. Unless stated otherwise, $+$ and $-$ on equal-length bit strings both denote XOR. The symbols $\boxplus$ and $\boxminus$ denote a stated group operation and its inverse. For a finite set S , the notation $x \leftarrow S$ denotes uniform sampling from S . Separate sampling operations use independent randomness unless stated otherwise.

We serialize a 128-bit block as sixteen bytes in AES/FIPS-197 order. Bytes $4c, \dots, 4c + 3$ form column c of the AES state. For integers $m \leq n$, let $[m, n] := \{m, m + 1, \dots, n\}$ and $[m, n) := \{m, m + 1, \dots, n - 1\}$. We abbreviate $[1, n]$ as $[n]$. For a set S , let $\mathcal{P}(S)$ denote its power set.

Let $\mathcal{D}$ and $\mathcal{Y}$ be finite sets with $\mathcal{Y} \neq \emptyset$. A uniform random function $F : \mathcal{D} \rightarrow \mathcal{Y}$ assigns an independent uniform value $F(x) \leftarrow \mathcal{Y}$ to each new input x and repeats that value on later queries. A random oracle has the same lazy-sampling interface on a specified domain, conventionally $\{0, 1\}^*$. We use *random-function behavior* for the concrete target on a fixed domain. We use *random-oracle model* when an application grants oracle access to the corresponding ideal function. Unless stated otherwise, each target applies to one specified family member and one fixed instance or epoch.

2.2 AES and AES-Round Designs

AES is a keyed permutation on 128-bit blocks [86, 3]. AES-128 expands its key into eleven round keys. After the initial AddRoundKey, nine full rounds apply SubBytes, ShiftRows, MixColumns, and AddRoundKey. The final round omits MixColumns. The AESENC instruction instead implements one full round for an explicitly supplied round key.

Our family uses the AES round as a public nonlinear map and supplies every round key directly. The three members change the input schedule or state width without changing the AES round. They therefore share the AES S-box, diffusion layer, and hardware instructions. They do not use the AES key schedule, final-round structure, or encryption interface.

Several other designs also use AES rounds directly. Simpira, Haraka, and Areion construct wide permutations or short-input hashes [54, 69, 31]. Haraka obtains a short-output hash by applying feed-forward after an AES-round permutation. Our wide construction instead injects the input after every layer. It retains the Haraka-256 v2 round constants and column permutation.

KIASU-BC is a keyed tweakable block cipher with an AES-like round [66, 102]. AEGIS and Tiaoxin use AES rounds in stateful authenticated-encryption

designs [101, 84, 68, 73]. Their interfaces and security targets differ from the public fixed-domain functions studied here. Attacks on Simpira nevertheless motivate explicit analysis of repeated dependencies [37, 95]. Jean found collisions for both widths of the first Haraka proposal in about 2^{16} work by exploiting symmetric round constants [67]. Haraka v2 changes those constants [69].

Published preimage attacks on Haraka-256 v2 reach 4.5 of its five rounds. Bao et al. give a classical attack costing 2^{224} [47], and Schrottenloher and Stevens give a quantum attack costing $2^{115.55}$ [93]. At the time of writing, we are not aware of a published shortcut collision, preimage, second-preimage, or partial-preimage attack on all five rounds of Haraka-256 v2. This statement concerns the hash interface and does not cover distinguishers on the underlying permutation. It also does not establish security. The designers make no collision-resistance claim for the five-round variant; their 128-bit collision-resistance claim uses six rounds [69].

For Haraka-512 v2, Bao et al. give a 2^{224} preimage attack on all five rounds [47]. Schrottenloher and Stevens give a full-round partial-preimage attack [93]: they prescribe 64 output bits and find a matching input using about 2^{32} time and 2^{32} memory, whereas generic search needs about 2^{64} evaluations. The result is neither a full preimage nor a collision.

Pairwise-independence results for AES-like ciphers with independent round keys provide a proof-oriented baseline [21]. They do not apply to our schedule, in which one input determines every round key. Designs that combine AES and SHA instructions provide a separate performance baseline [92].

2.3 Short-Input Hashing

For an ideal cipher E and fixed plaintext p , the map $x \mapsto E(x, p)$ is a uniform random function because distinct keys select independent permutation tables. Variable-key AES is the direct concrete analogue, but it expands a new key for each input. The base construction seeks the same one-block interface without that expansion.

PGV compression functions wire a block cipher with XOR feed-forward [9, 10]. Their collision and preimage guarantees do not imply random-oracle indistinguishability when an adversary also sees the primitive and its inverse; rate-one examples such as Davies–Meyer fail in that interface [11]. Linicrypt characterizes broader algebraic constructions under other ideal-cipher interfaces [12, 13].

Rogaway and Steinberger prove collision and preimage bounds for compression functions built from linear combinations of message blocks and permutation outputs, including constructions that reuse one permutation [105]. Their LP framework is a structural predecessor to our affine wrapper. Their bounds concern different, compressing maps and do not establish the length-preserving random-function interface targeted here.

Permutation feed-forward is widely used for correlation-robust hashing in MPC [59, 44, 33]. Recent work extends this line to multi-user tweakable correlation robustness under key-leakage queries and applies the result to OT extension [60]. These results target specified input correlations, rather than a general

random-oracle interface with access to the underlying permutation. Stronger random-oracle behavior from public permutations usually costs additional primitive calls. In particular, the sum of two independent public permutations has a long sequence of indistinguishability analyses [82, 19, 61, 20].

Sponge and Grøstl-style constructions provide variable-length alternatives with wider states or multiple permutations [23, 6, 53, 2, 76]. Those architectures solve a broader problem than the fixed-domain interfaces studied here.

3 The Base Construction

The base construction maps one 128-bit block to one 128-bit block. Its input may be correlated with secret offsets held elsewhere in an MPC protocol. The target is therefore a uniform random function on the one-block domain, not a keyed PRF or an encryption scheme.

Let $R : \{0, 1\}^{128} \rightarrow \{0, 1\}^{128}$ be the keyless part of a full AES round: SubBytes, ShiftRows, and MixColumns. An AESENC instruction computes $R(q) + k$ for state q and round key k . Each round key is the input x XORed with a public constant. The eleven constants $c_1, \dots, c_{11}$ are fixed; their deterministic derivation appears below.

Routine $H_{128}(x)$:

Upon input $x \in \{0, 1\}^{128}$, set $q_1 := x + c_1$. For $j \in \{1, \dots, 10\}$, compute

$$r_j := R(q_j), \quad q_{j+1} := r_j + x + c_{j+1}.$$

Output q_{11} .

Figure 1: The construction $H_{128} : \{0, 1\}^{128} \rightarrow \{0, 1\}^{128}$. It uses eleven public constants $c_1, \dots, c_{11}$.

Thus one evaluation consists of an initial XOR followed by ten AESENC-style rounds. Every round includes MixColumns; there is no AESENCLAST-style final round. Unlike variable-key AES, the construction performs no key expansion. Unlike fixed-key AES, it is not designed or analyzed as a permutation. Its target random-function interface permits generic collisions near the birthday bound.

The construction replaces AES key expansion with the input-tied schedule

$$K_i(x) := x + \bar{c}_i \quad (i = 0, \dots, 10),$$

where $\bar{c}_i := c_{i+1}$. We call the schedule input-tied because the same input x determines every round key; only the public offset varies with i . This schedule enables direct AES instructions but also exposes structured related-key differences. View one evaluation as AES-style encryption of the fixed plaintext $p = 0$ under round keys $K_0, \dots, K_{10}$. Two inputs x, x' then induce

$$\Delta p = 0, \quad \Delta K_0 = \dots = \Delta K_{10} = \Delta x.$$

Section 6 analyzes this exact interface.

3.1 Round Count and Attack Reach

The number of rounds is an engineering parameter, not a consequence of a security theorem. For a construction with ρ rounds, let r_d be the greatest depth reached by a demonstrated attack on the same one-block interface. The attack must distinguish the construction with stated resources or beat the relevant generic collision or preimage bound.

Let $r_c \geq r_d$ be the greatest depth reached when explicit attack candidates are also counted. An attack candidate has a specified procedure and cost model, but incomplete validation. We report both depths. The quantities $\rho - r_d$ and $\rho - r_c$ give the round distances from the full construction to demonstrated-attack and attack-candidate reach, respectively. These distances are not security bounds. They omit time, data, memory, success probability, and attack model, which we report separately.

For the 128-bit domain and range in this paper, generic collision search takes about 2^{64} evaluations, while fixed-target preimage search takes about 2^{128} . A structural trail or model-specific exclusion contributes evidence, but it does not determine either reach depth unless it yields a corresponding attack or attack candidate below the relevant generic baseline. This distinction is important here because several of our deeper results exclude only restricted attack models.

Round distance is a design reference, as in the Rijndael and Keccak rationales, but attack complexity and access model remain separate considerations [87, 4, 7].

We select $\rho = 10$ for two reasons. First, ten full rounds match the round depth and total round-instruction count of AES-128. The construction uses ten AESENC instructions; AES-128 uses nine AESENC instructions and one AES-ENCLAST instruction. More rounds would increase cost; fewer rounds would reduce the distances to the results below. Second, the deepest attacks in Section 6 leave a distance of at least five rounds. The analytic integral distinguishers give $r_d = 3$. The five-round anchored MITM procedure is an attack candidate with estimated time 2^{120} ; its cost model predicts an eight-bit saving over generic fixed-target preimage search, so $r_c = 5$. The demonstrated-attack distance is therefore seven rounds. We use the five-round attack-candidate distance as the round-count reference, not as a proved security margin. Every state byte also depends structurally on all input bytes after two forward or two inverse rounds. The reduced-round differential models give active-S-box minima comparable to those of fixed-key AES.

Table 1 lists the principal attack classes that these analyses do not cover. The ideal-permutation theorem in Section 7 does not determine the round count.

3.2 Public Constants and Instances

For interoperable implementations, we derive the constants from a domain-separated extendable-output function (XOF). Let $a \in [0, 2^{64})$ be a public in-

stance identifier, let $j \in [1, 2^{32})$ be a round index, and fix the domain separator

$$D_{128} := 616573486173682f487031302d763100.$$

The displayed hexadecimal string lists the bytes in transmission order. For every instance a , set

$$c_j^{(a)} := \text{SHAKE256}(D_{128} \parallel \text{LE64}(a) \parallel \text{LE32}(j), 16) \quad (j = 1, \dots, 11).$$

The final argument requests 16 output bytes [5]. The notation $H_{128,a}$ denotes the construction with schedule $(c_1^{(a)}, \dots, c_{11}^{(a)})$. The default instance has $a = 0$; we write $H_{128} := H_{128,0}$ and $c_j := c_j^{(0)}$. Its first constant is

$$c_1 = \text{d4e69911444f5403cdd33decab71e2c0}.$$

The XOF derivation deterministically assigns a constant to each encoded instance-and-round pair. The fixed-length encodings give distinct pairs distinct XOF inputs.

The implementation identifier is H128. The vector $(c_1, \dots, c_{11})$ is part of this profile; the interface does not accept independently chosen round constants. This restriction is material. A parameter generator that controls the penultimate constant can program one branch before the last AES round and plant a collision. [Appendix E](#) gives exact examples. The XOF derivation binds each constant to the fixed domain separator and instance identifier. It does not assert security for arbitrary constant vectors.

[Appendix D](#) gives test vectors for the default instance. The portable reference code and machine-readable vectors accompany the paper. [Table 1](#) states the construction-level target. [Section 6](#) gives the concrete cryptanalysis.

4 Tweakable and Wide Constructions

This section specifies the other two family members. The tweakable construction supports a public counter stream by splitting the counter into an eight-bit tweak and a 64-bit epoch. The wide construction accepts two complete input blocks and returns 256 bits. Both retain the input-fed design principle. [Section 6.9](#) evaluates the new structure introduced by each construction.

4.1 The Tweakable Construction

A public tweak distinguishes uses of the same message within one epoch. The construction takes a message $m \in \{0, 1\}^{128}$ and a tweak $i \in [0, 256)$. A public epoch $a \in [0, 2^{64})$ selects the constant schedule. For each epoch, define

$$H_{128,a}^{\text{tw}} : \{0, 1\}^{128} \times [0, 256) \longrightarrow \{0, 1\}^{128}.$$

For a fixed epoch, the target is a uniform random function on this paired domain. Equivalently, its 256 tweak slices are independent in the ideal interface. Both

a and i are public and contribute no secret entropy. We write H_{128}^{tw} for epoch $a = 0$. Epoch selection is external; the function itself neither wraps the tweak nor increments the epoch.

Each epoch uses eleven base constants from a fixed public stream. The tweak adds one mask at each boundary. We define the stream, the masks, and then the round computation.

Let

$$D_{\text{key}} := 616573486173682f4831323874772f6165732d6b657900$$

be the ASCII string `aesHash/H128tw/aes-key` followed by one zero byte. Define and publish the AES-128 key

$$K_{\text{tw}} := \text{SHAKE256}(D_{\text{key}}, 16) = \text{a5df71aff064d44dbbc6bde5dd1fc1b8}.$$

The final argument requests 16 output bytes [5]. Implementations embed the displayed key; they do not invoke SHAKE256 at runtime. For $n \in [0, 2^{128})$, define the AES-counter stream

$$S_n := \text{AES}_{K_{\text{tw}}}(\text{LE128}(n)).$$

Epoch a uses eleven consecutive stream blocks:

$$b_j^{(a)} := S_{11a+j-1}, \quad j = 1, \dots, 11.$$

The largest addressed counter is below 2^{68} . Thus the supported epochs use disjoint counter intervals without wraparound. Because AES is a permutation, all addressed base constants are distinct. The stream and its key are fixed parts of the profile; callers select only the epoch.

To define the tweak masks, let

$$\mathbb{F} = \mathbb{F}_2[z]/(z^{128} + z^7 + z^2 + z + 1).$$

Encode a field element as a little-endian 128-bit integer: bit b is the coefficient of z^b . Define $I(i) := \text{LE32}(i) \parallel 0^{96}$. Fix the nonzero multiplier

$$\beta := \text{e5fe6308550d34e64f5d5f579c698fcb}.$$

The displayed bytes use the same field encoding. Although the tweak has only eight bits, the four-byte embedding fixes its serialization. Set

$$t_1(i) := \beta I(i), \quad t_{j+1}(i) := z t_j(i) \quad (1 \leq j \leq 10).$$

On input (m, i) , initialize

$$q_1 := m + b_1^{(a)} + t_1(i).$$

For $j = 1, \dots, 10$, compute

$$q_{j+1} := R(q_j) + m + b_{j+1}^{(a)} + t_{j+1}(i),$$

and return q_{11} . The implementation identifier is **H128tw**.

For a fixed epoch and tweak, this is the H_{128} recurrence with translated public constants. For two inputs (m, i) and (m', i') in the same epoch, write $\Delta m = m + m'$ and $\Delta i = I(i) + I(i')$. Boundary j receives

$$d_j := \Delta m + \beta z^{j-1} \Delta i. \quad (1)$$

If $i \neq i'$, at most one of $d_1, \dots, d_{11}$ is zero. Indeed, two zero values at positions $j < k$ would imply

$$\beta(z^{j-1} + z^{k-1})\Delta i = 0.$$

The embedding I is injective, so $\Delta i \neq 0$. The fixed value β is also nonzero. Finally, $z^{j-1} + z^{k-1}$ is represented by a nonzero polynomial of degree below 128. Thus all three factors are nonzero in $\mathbb{F}$, which gives a contradiction.

The construction uses ten rounds, matching the base construction. **Section 6.9** evaluates the tweak-slice schedules and the relations introduced across tweak slices.

Counter streams. MPC protocols commonly hash messages under successive public positions $u = 0, 1, 2, \dots$. For $u \in [0, 2^{72})$, split the counter as

$$i(u) := u \bmod 2^8, \quad a(u) := \lfloor u/2^8 \rfloor.$$

A caller hashes position u by evaluating $H_{128,a(u)}^{\text{tw}}(m, i(u))$. The low byte selects the tweak, and the upper 64 bits select the base schedule. Hence the eight-bit tweak does not limit the position space. It sets the refresh interval: one schedule serves 256 consecutive hashes.

For $a < 2^{64} - 1$, incrementing u from $256a + 255$ to $256(a+1)$ selects the next eleven stream blocks and resets the tweak to zero. A sequential implementation continues the AES-counter stream. A random-access implementation starts at counter $11a$.

Constant generation and field arithmetic can be confined to epoch or cache-window setup. The fixed key schedule is expanded once. The hash path then loads eleven effective constants and performs ten AESENC instructions. Thus the online round-instruction count matches one AES-128 evaluation per hash; this comparison excludes epoch setup. **Appendix D** gives the cache layout and test vectors. The portable and unrolled AES-NI implementations agree on those vectors.

4.2 The Wide Construction

The wide construction maps two input blocks to two output blocks:

$$H_{256} : \{0, 1\}^{256} \longrightarrow \{0, 1\}^{256}.$$

Its target is a uniform random function on this domain and range. Split an input as $X = x_0 \| x_1$, where each $x_\ell \in \{0, 1\}^{128}$. The construction maintains two 128-bit lanes in AES byte order and initializes

$$(s_0, s_1) := (x_0 + \text{iv}_0, x_1 + \text{iv}_1),$$

where the fixed initialization constants are

$$\begin{aligned} \text{iv}_0 &= 2419563\text{e}0\text{f}221\text{bf}2\text{de}86\text{f}28\text{a}32\text{a}7\text{deed}, \\ \text{iv}_1 &= 5245\text{fb}1\text{cb}0\text{c}05\text{dbe}1\text{febece}1\text{ada}61032. \end{aligned}$$

The ten layers use the first twenty Haraka-256 v2 round constants [69]; [Appendix D](#) fixes their byte strings.

Write each lane as four consecutive 32-bit columns,

$$s_0 = w_0 \| w_1 \| w_2 \| w_3, \quad s_1 = v_0 \| v_1 \| v_2 \| v_3,$$

and define the column permutation

$$\text{mix}_{256}(s_0, s_1) := (w_0 \| v_0 \| w_1 \| v_1, w_2 \| v_2 \| w_3 \| v_3).$$

For $j = 0, \dots, 9$, compute

$$y_\ell := R(s_\ell) + x_\ell + \text{RC}_{2j+\ell}, \quad \ell \in \{0, 1\}.$$

If j is odd, set $(s_0, s_1) := \text{mix}_{256}(y_0, y_1)$; otherwise set $(s_0, s_1) := (y_0, y_1)$. Thus mixing follows layers 2, 4, 6, 8, and 10. After the final mixing step, define

$$H_{256}(X) := s_0 \| s_1.$$

The implementation identifier is H256. The construction has no public instance identifier and applies no additional feed-forward, padding, length encoding, or finalizer.

An application that needs a 128-bit digest may truncate $H_{256}(X)$ to its first 128 bits. This operation is ordinary output truncation, not a separate family member. Under the random-function target for H_{256} , truncation gives a uniform random function from 256 bits to 128 bits. Its ideal collision and fixed-target preimage costs are therefore the generic costs for a 128-bit range.

The portable and AES-NI implementations agree on the complete-state vectors in [Appendix D](#). The construction uses twenty AESENC evaluations at depth ten. Each input block enters all ten layers, matching the depth selected for the base construction. This is a design rule, not a derived round bound. [Section 6.9](#) evaluates partial-preimage attacks and the collision, dependency, slide, rebound, and repeated-permutation mechanisms relevant to the two-lane state.

5 Implementation and Performance

This section measures the cost of H_{128} relative to fixed-key and variable-key AES-128. It also measures generation of the AES-counter stream for H_{128}^{tw} . The wide construction is cross-checked against portable references but is not benchmarked.

The AES-NI implementation maps each construction round to one x86 AES-ENC instruction. In the notation of [Section 3](#),

$$q_1 := x + c_1, \quad q_{j+1} := \text{AESENC}(q_j, x + c_{j+1}).$$

The implementation uses fixed-width kernels with ten explicitly unrolled rounds. The scalar kernel measures dependent-call latency. The batched kernels maintain eight independent AES-NI lanes or sixteen independent blocks across four 512-bit VAES vectors. Every executable checks the published test vectors before measuring performance.

We benchmarked on one core of an AMD Ryzen 9 7950X with frequency boost disabled. The host ran Ubuntu with Linux kernel 6.17.0-41 and GCC 15.2. Each trial used one million dependent kernel invocations, and all benchmark processes ran serially. [Table 2](#) reports the median of seven trials. The unit is invariant-TSC ticks per 128-bit block, so comparisons are meaningful within this machine but are not portable cycle counts.

Kernel	Parallel blocks	TSC ticks/block
H_{128} , AES-NI latency	1	41.44
Fixed-key AES-128 latency	1	41.14
Variable-key AES-128 latency	1	83.17
H_{128} , AES-NI batch	8	5.19
Fixed-key AES-128, AES-NI batch	8	5.14
H_{128} , VAES batch	16	2.63
Fixed-key AES-128, VAES batch	16	2.57

Table 2: Median performance on the Ryzen 9 7950X. Fixed-key AES-128 uses a pre-expanded key. Variable-key AES-128 includes AES-128 key expansion for each block.

The H_{128} latency is within one percent of fixed-key AES-128 on this host. Its eight-lane AES-NI throughput is also within one percent. The sixteen-block VAES kernel is 2.3% slower than the corresponding fixed-key AES kernel. In contrast, AES-128 with a fresh input-derived key and key expansion takes approximately twice the latency of H_{128} . These results support the performance claim: the construction costs approximately one full AES evaluation while avoiding per-input key expansion.

For H_{128}^{tw} , one epoch requires eleven fixed-key AES-128 evaluations and serves 256 positions. The benchmark computes the random-access starting counter 11a and writes all 176 schedule bytes. It expands the fixed AES key before timing. The eleven-lane AES-NI generator takes 63.27 TSC ticks, or 4.8% of the cost of 256 batched AES-NI H_{128} evaluations. The twelve-lane VAES generator discards one block and takes 46.41 ticks, or 6.9% of the cost of 256 batched VAES H_{128} evaluations.

The **benchmarks** directory contains the source, build flags, raw trials, CPU configuration, and median checker.

6 Concrete Cryptanalysis

We first analyze the base construction against its random-function target, then examine the relations introduced by tweaks and the two-lane state. For the base construction, the attack interface can be written in AES notation. For $i = 0, \dots, 10$, define

$$K_i(x) := x + \bar{c}_i, \quad K(x) := (K_0(x), \dots, K_{10}(x)),$$

where $\bar{c}_i := c_{i+1}$. Let $\text{AES}_{K(x)}^*(0)$ apply the initial key $K_0(x)$ to the fixed plaintext zero, followed by ten full AES rounds with keys $K_1(x), \dots, K_{10}(x)$. Then

$$H_{128}(x) = \text{AES}_{K(x)}^*(0).$$

For $r \in [1, 10]$, let $H_{128}^{[r]}$ denote the same recurrence stopped after r AESENC-style rounds, including the boundary injection after round r . The notation AES^* does not use the AES key schedule. Every round includes SubBytes, ShiftRows, and MixColumns; none uses the AESENCLAST instruction. Because the plaintext is fixed and every round key is $x + \bar{c}_i$, two inputs x, x' induce

$$\Delta p = 0, \quad \Delta K_0 = \Delta K_1 = \dots = \Delta K_{10} = \Delta x.$$

The attacker may choose x and evaluate the resulting public function. It cannot vary the plaintext or any round key independently. This input-tied interface is the concrete attack surface analyzed below.

Table 3 gives a global view before the individual attack analyses. The integral analysis proves a three-round distinguisher. The anchored meet-in-the-middle (MITM) analysis gives a five-round candidate with estimated time 2^{120} . Results at greater depth are restricted-model exclusions or incomplete searches, not full-round security bounds.

Technique	Coverage	Result and scope
Meet-in-the-middle analysis	Five-round candidates; model exclusion through ten	No anchored coloring for rounds 6–10; guessed values and superposition states remain outside the model
Truncated related-key differentials	Three-round certified minima; one sparse 10-round class	Certified truncated minima 1, 4, 11 through three rounds; the 10-round result excludes only trails with at most one active S-box per round
Collision-compatible trails	Three-round certified minima; incomplete deeper searches	Certified truncated minima 8, 15, 16 through three rounds; the deeper searches establish no full-round bound
Linear-mask cancellation	One complete round and seven selected two-round hulls	Best one-round mask found has correlation magnitude $2^{-31.256}$; selected two-round correlations have magnitude at most $2^{-46.810}$
Integral and division property	Three rounds	Fifty-four two-byte cubes balance all 16 output bytes for all constants and inactive bytes, using 2^{16} queries
Cross-instance trails	Ten rounds in a surrogate model	No collision-compatible trail with at most 25 active S-boxes in the all-active model; not a general multi-instance bound
Fixed-key AES comparison	Three rounds	Truncated minima 1, 4, 11 versus 1, 5, 9; no round-for-round security ordering

Table 3: Reduced-round evidence and model boundaries. No row gives a full-round attack or proves full-round security.

6.1 Attack Goals and Models

We consider differential, linear, integral, and MITM cryptanalysis under the input-tied interface. A *differential trail* describes how an XOR difference propagates through the internal state. For a block cipher, the usual goal is an output difference whose probability exceeds its generic value. A collision-compatible trail for this hash must additionally end with zero output difference:

$$\Delta p = 0, \quad \Delta K_i = \Delta x, \quad \Delta H_{128} = 0.$$

Such a trail is analogous to a fixed-target-plaintext key collision for AES, as studied by Taiyama et al. [94], except that our round-key differences are all tied to the single value Δx .

Integral, or square-style, distinguishers originated with the SQUARE block cipher [41] and were later formalized as integral cryptanalysis [72]. AES attacks evaluate structured plaintext sets under a fixed, typically secret, key and track balanced output bytes [45, 71, 34]. Here the plaintext is fixed, while every round key varies with x . We therefore adapt the integral method to the resulting input-as-round-key circuit. Classical AES attacks often add rounds by guessing key bytes and partially decrypting [45, 34]. Our interface has no secret AES key, so the endpoint is a direct random-function distinguisher.

We abbreviate Matyas–Meyer–Oseas, Miyaguchi–Preneel, and Davies–Meyer as MMO, MP, and DM.

Work	Adversarial freedom	Relation to H_{128}
AES pseudo-preimages [47, 35]	The pseudo-preimage setting controls both the chaining value and message; a fixed-key variant controls the plaintext.	Our interface has one 128-bit variable x , a fixed plaintext, and eleven round keys tied to x .
AES collisions and one-way functions [52]	MMO/MP fixes the initial key and varies the message. DM fixes the plaintext and varies a key expanded by the AES schedule.	The DM attack has the closest endpoint, but our schedule replaces AES expansion by the values $x + \bar{c}_i$.
Triangulating rebound [36]	The semi-free-start setting supplies independent state and message degrees of freedom.	Those degrees of freedom are absent from the one-input hash interface.
Haraka and Simpira [67, 37, 95]	The attacks exploit wide-state mixing, Feistel dependencies, or structured round constants.	They do not transfer directly, but they warn against counting nominally distinct active transitions as independent.

Table 4: Attack interfaces in the closest AES-hashing literature. A round count transfers only after the construction’s degrees of freedom and key relations have been preserved.

Meet-in-the-middle preimage cryptanalysis is another central line of work on AES-like hashing. Bao et al. automate the search and reach eight of ten rounds of AES-128 hashing, as well as Haraka targets [47]. Superposition states and later refinements extend the framework [50, 51]. Degré et al. recover the eight-round AES-128 result with a simpler model that makes validity conditions explicit and reduces its memory cost [35]. Chen et al. give the closest published collision setting: a classical one-block collision attack on seven-round AES-MMO/MP and a fixed-target-plaintext key collision on four-round AES-128-DM [52]. Triangulating rebound attacks reach seven-round semi-free-start AES-MMO/MP collisions and include a practical six-round instance [36].

6.2 Anchored Meet-in-the-Middle Analysis

The objective is preimage search: given an output target y , find an input x that produces it. A MITM attack tries to replace full evaluations of candidate inputs with two lists of partial computations. One list proceeds forward from the input; the other proceeds backward from y . The computations meet at an intermediate round boundary, called the *cut*. Matching selected state bytes at the cut produces candidate inputs that must still be checked against the full target. Here the two lists are *anchored* at the input and output boundaries.

We partition the unknown input bytes by their role in the search. *Blue* bytes vary in the forward list, *red* bytes vary in the backward list, and *gray* bytes are fixed while those lists are processed. The search repeats over gray assignments. For example, the five-round candidate below uses five blue bytes, two red bytes, and nine gray bytes. The colors describe how the attacker organizes guesses; they do not change the construction.

Because x is injected repeatedly, disjoint varying bytes do not make the two lists independent: forward states can depend on red bytes, and backward

states can depend on blue bytes. The constrained search groups assignments into *buckets* that hold selected intermediate quantities fixed. Within a pair of buckets, this can let each list compute the match bytes independently of the other list’s precise assignment. We first test the plain split, then introduce these constraints and reconstruct the five-round candidate.

Throughout this subsection, write s_j for the state after j rounds: $s_0 = x + \bar{c}_0$ and $s_j = q_{j+1}$ in the construction’s notation. Inverse rounds compute $s_{j-1} = R^{-1}(s_j + x + \bar{c}_j)$ for a fixed or guessed x ; they do not give inverse hash queries. Backward states still depend on x and must reach $s_0 = x + \bar{c}_0$ for a valid preimage.

The plain-split test exhausts all complementary partitions of the sixteen input bytes into forward and backward neutral sets. At every round boundary, the dependency analysis tracks which input bytes determine each state byte. SubBytes preserves each dependency set, ShiftRows permutes the sets, and MixColumns takes their column-wise union. AddRoundKey adds the corresponding byte of x .

After two forward rounds, every propagated dependency set contains all sixteen bytes of x . Backward propagation from a fixed target, including the injected bytes of x , has the same property after two inverse rounds. Consequently, at every internal cut of the ten-round construction, exhaustive search finds no byte that the two sides can compute from disjoint nonempty subsets of x . This rules out the plain two-list neutral-byte split without guessed intermediate values.

We next adapt the constrained-neutral-word model of Bao et al. [47]. The two lists start at $s_0 = x + \bar{c}_0$ and $s_\rho = y$. The model reuses each byte’s color at $s_0[j]$ and every AddRoundKey node, so it counts each neutral byte once. It applies the XOR⁺, MC⁺, and combined XOR-MC[−] rules of Bao et al. A constraint consumes one byte of freedom when it removes the opposite list’s influence. For a candidate with d_+ forward bytes, d_- backward bytes, and an m -byte match, define

$$v := \min\{d_+, d_-, m\}.$$

The standard list analysis gives time 2^{128-8v} . Table 5 records candidates rather than optimal attacks.

The nonlinear validation is stronger than the color search but has limited scope. For five rounds, it fixes the nine gray bytes to zero and covers two fixed output targets. These checks validate the split maps at those settings; they do not establish a success distribution over all gray assignments and targets. Appendix A gives the four- and five-round split maps, their exhaustive checks, and the observed fiber sizes.

Reconstructing the five-round candidate. The five-round candidate uses the following byte partition, in AES column-major order:

$$\mathcal{B} = \{0, 1, 2, 5, 7\}, \quad \mathcal{R} = \{11, 12\}, \quad \mathcal{G} = \{3, 4, 6, 8, 9, 10, 13, 14, 15\}.$$

The sets $\mathcal{B}$, $\mathcal{R}$, and $\mathcal{G}$ contain the forward-neutral, backward-neutral, and fixed bytes, respectively. The cut is s_2 . Forward propagation determines bytes

Rounds	(d_+, d_-, m)	Standard list cost	Validation
2	(5, 5, 5)	2^{88} time; 2^{40} entries	Six independent linear constraints and a five-byte match
3	(2, 2, 1)	2^{120} time; 2^{16} entries	Four independent constraints and a one-byte match
4	(1, 1, 3)	2^{120} time; 2^8 online entries	Nonlinear fiber checks for two targets
5	(1, 1, 1)	2^{120} time; 2^8 average online entries	Split-map checks for two targets and an explicit cut-byte evaluator
6–10	none	No saving in this model	UNSAT at every internal cut for $v \geq 1$

Table 5: Anchored MITM analysis for the input-tied $x + \bar{c}_i$ schedule. The displayed memory is the online list size and excludes storage used to construct the nonlinear initial structures.

$\{4, 5, 6, 7\}$ at the cut, and backward propagation determines bytes $\{1, 6, 11, 12\}$. Thus the two lists match on $s_2[6]$.

Outline of the five-round anchored MITM candidate. All byte arithmetic below is in the AES field, and S is the AES S-box.

1. Fix the nine gray bytes in $\mathcal{G}$ and the target s_5 .
2. Bucket the 2^{16} assignments to $\mathcal{R}$ by $\eta_{\mathcal{R}} := S(x[12] + \bar{c}_0[12]) + 03 \cdot S(x[11] + \bar{c}_0[11])$. Each bucket has exactly 2^8 assignments.
3. Symbolically invert the last round from s_5 and form $u_3 := \text{MC}^{-1}(s_4 + x + \bar{c}_4)$. Bucket the 2^{40} blue assignments by the four-byte bucket label $\eta_{\mathcal{B}}$ defined in [Appendix A.7](#). Within each bucket, $u_3[3], u_3[6], u_3[9], u_3[12]$ are independent of the blue representative.
4. For each pair of nonempty buckets, evaluate the blue assignments forward to $s_2[6]$ and the red assignments backward to $s_2[6]$. Match the two lists, combine each matching pair into x , and verify the full target.

Figure 2: Outline of the five-round procedure. The constraints make the forward cut byte independent of the red representative and the backward cut byte independent of the blue representative.

For fixed gray bytes, the four blue-side constraints split into maps on 24 and 16 input bits. There are 2^{40} blue assignments and 2^{16} red assignments. Matching on one byte gives the standard cost $2^{40+16-8} = 2^{48}$ for each gray assignment. Repeating over the 2^{72} gray assignments gives the 2^{120} time estimate. Constructing the blue buckets costs 2^{40} operations per gray assignment, or 2^{112} in total. This setup is below the list-processing cost, although its storage is not included in the 2^8 average online-list figure in [Table 5](#).

The six-round cutoff is a statement about this anchored color model. The model does not include guessed intermediate values, superposition states, or an internal initial structure whose relation to the same x has been proved. Extending the analysis to those models, or to a two-copy collision model, remains open [\[50, 35\]](#).

The remaining analysis uses automated and model-based methods for the

stated reduced-round attack classes.

6.3 Truncated Related-Key Differentials

This subsection asks whether the input-tied round keys admit unusually sparse related-key differential trails. A truncated trail records only whether each byte difference is zero or nonzero; a nonzero byte is *active*. The number of active S-boxes is a standard screening metric because the maximum differential probability of the AES S-box is 2^{-6} . Here it is only a structural metric: the repeated key difference creates dependencies that preclude multiplying per-S-box probabilities without further analysis.

Our model follows prior constraint-programming (CP) and mixed-integer linear programming (MILP) searches for AES differential trails and related-key schedules [58, 22, 32]. It fixes the plaintext difference to zero and injects the same nonzero difference $\Delta K_i = \Delta x$ in every round. This input-tied constraint is the essential difference from the fixed-key AES calibration, where $\Delta K_i = 0$ and the plaintext carries the input difference. [Appendix A](#) gives the complete activity encoding.

The model is conservative. An unsatisfiability certificate excludes every byte-level characteristic with the specified activity bound. A satisfying activity pattern need not correspond to a characteristic once exact S-box differences and linear equations are imposed.

For one, two, and three rounds, the input-tied model proves minimum active-S-box counts of 1, 4, and 11. The corresponding fixed-key AES minima in the same model are 1, 5, and 9. The full ten-round model also excludes trails having at most one active S-box in each round. These coarse counts give no round-for-round ordering with fixed-key AES; collision-compatible trails impose the more relevant condition considered next.

6.4 Collision-Compatible Trails

An ordinary low-activity trail is not yet a hash collision: it must also have zero terminal output difference. We add this constraint to the same truncated model. The resulting reduced-round minima are substantially larger than the unrestricted ones: 8, 15, and 16 active S-boxes for one, two, and three rounds. The solver certified all three values. For four rounds it found a 19-active trail but did not certify optimality.

Rounds	Active S-boxes	Certificate
1	8	proved minimum
2	15	proved minimum
3	16	proved minimum
4	19	best found in 300 seconds

Table 6: Collision-compatible truncated related-key differential trails for the AES-derived hash model. “Proved minimum” means that the CP solver certified optimality in the truncated model.

Collision compatibility is the restriction relevant to the hash. A collision-compatible trail must respect $\Delta K_i = \Delta x$ in every round and return to zero output difference, while the interface supplies only the encryption of the fixed plaintext zero under each input-dependent key. Thus an ordinary related-key AES distinguisher does not automatically yield a distinguisher or collision for the hash. We have no certified full-round lower bound; the inconclusive threshold run and its limits are recorded in [Appendix A](#).

6.5 Cross-Instance Trails

The many-instance setting introduces a second question. As a surrogate for two domain-separated schedules, independently sample $\bar{c}_i, \bar{d}_i \leftarrow \{0, 1\}^{128}$ for $i = 0, \dots, 10$. Define $\bar{C} := (\bar{c}_0, \dots, \bar{c}_{10})$ and $\bar{D} := (\bar{d}_0, \dots, \bar{d}_{10})$. Write $H_{\bar{C}}$ and $H_{\bar{D}}$ for the base recurrence under these schedules. Comparing $H_{\bar{C}}(x)$ with $H_{\bar{D}}(x')$ gives

$$\Delta K_j = \Delta x + \bar{c}_j + \bar{d}_j.$$

Thus independent public offsets replace the same-instance relation $\Delta K_j = \Delta x$. The adversary sees these offsets but does not choose them, unlike a related-key adversary that may request freely related keys.

We restrict the model to trails for which every byte of every ΔK_j is active. For a fixed Δx independent of the sampled schedules, this event has probability $(255/256)^{176} \approx 0.50$ over the schedule sampling. This probability does not apply to differences chosen after the schedules are revealed. The restriction therefore supplies no unconditional cross-instance bound. In this model, the certified collision-compatible minima for one, two, and three rounds are 16, 20, and 23 active S-boxes; a four-round search found 27 without proving optimality. The full-round threshold instance with at most 25 active S-boxes is unsatisfiable. [Appendix A](#) records the complete table and solver status.

The model assumes independent uniform offsets, and its certified minima exceed the corresponding same-instance minima. Correlated schedules may instead make $\bar{c}_j + \bar{d}_j$ constant or low-dimensional across rounds, collapsing the comparison toward $\Delta K_j = \Delta x$; the model does not cover such schedules.

6.6 Linear-Mask Cancellation Checks

Linear cryptanalysis studies biased relations between masks on the input and output of a cipher [78]. The repeated copies of x create a different mask constraint from fixed-key AES. We make that constraint explicit before describing the search.

Write the round function as $R = L \circ S$, where S applies the 16 AES S-boxes and L applies ShiftRows followed by MixColumns. Consider the construction stopped after r rounds. Let λ_i be the mask on q_i , and let ω be the mask on the output q_{r+1} . Set $\mu_i := \lambda_{i+1}$ for $i < r$ and $\mu_r := \omega$. The S-box output mask in round i is $\nu_i := L^\top \mu_i$. The AES linear-approximation table (LAT) determines the correlation of each S-box transition from λ_i to ν_i .

The input x occurs in q_1 and after every round. Therefore, the external input mask of the trail is

$$\alpha = \lambda_1 + \lambda_2 + \cdots + \lambda_r + \omega.$$

We search for $\alpha = 0$ and $\omega \neq 0$. Such a trail contributes to an output-only linear relation. The public constants change each trail’s sign, but not its correlation magnitude.

An activity model and an exact-mask SAT model certify that trails compatible with cancellation need at least 8, 15, 16, and 25 active S-boxes through one, two, three, and four rounds. The exact model uses the AES LAT, so it also records each trail’s signed correlation. [Appendix A](#) gives the encoding, certificates, and witness weights.

For one round, $\alpha = 0$ forces $\lambda_1 = \omega$, and the output mask fixes every S-box transition. The trail product is therefore the complete component correlation, up to the sign contributed by the public constants. The best mask found has magnitude $2^{-31.256}$, giving a standard sampling complexity of about $2^{62.512}$. This is a sub-birthday distinguisher for the one-round function.

For two or more rounds, trails with the same external masks form a linear hull [79]; a single trail does not determine the hull correlation [77]. For

$$C_\omega := \mathbb{E}_{x \leftarrow \{0,1\}^{128}} \left[(-1)^{\langle \omega, H_{128}^{[2]}(x) \rangle} \right],$$

we compute complete hulls for seven selected output masks. All seven exhibit substantial sign cancellation. The strongest has $|C_\omega| = 2^{-46.810}$, corresponding to about $2^{93.620}$ evaluations. The selection search is bounded and does not maximize $|C_\omega|$ over every mask. These calculations make no claim for three or more rounds or for the ten-round construction; [Appendix A](#) records the masks, selection procedure, and complete contractions.

6.7 Integral and Square-Style Checks

We complement the differential search with a division-property model for integral attacks. Todo introduced the division property as a generalization of integral propagation [96]; the bit-based refinement tracks individual output bits [97].

Automated MILP and SAT searches for division trails are now standard tools for finding integral distinguishers [98, 99]. At a high level, an integral attack evaluates a function on a structured input set, often called a cube. It then asks whether an output bit or byte sums to zero. In our model, unsatisfiability for a chosen cube and output bit proves that the bit is balanced. Satisfiability only means that the model does not prove balance.

The classical Square attack balances all output bytes of three-round AES over a set with one varying byte [41, 72, 45, 71, 34]. Repeated input injection disrupts that propagation argument here; the next proposition instead varies two bytes. The repository gives the longer comparison [1]. The output of $H_{128}^{[3]}$ is q_4 . Write an AES state byte as $x_{r,c}$, where $r, c \in \{0, 1, 2, 3\}$ are its row and column.

Proposition 6.1. *Let $U = (r, c)$ and $V = (s, d)$ be distinct input positions. Vary $A := x_{r,c}$ and $B := x_{s,d}$ over $\{0, 1\}^8$, while holding the other 14 input bytes fixed. Define their columns after ShiftRows as*

$$u := c - r \pmod{4} \quad \text{and} \quad v := d - s \pmod{4},$$

and assume $u \neq v$. Let

$$E := E_u \cup E_v, \quad E_u := \begin{cases} \{u\}, & c = v, \\ \emptyset, & c \neq v, \end{cases} \quad E_v := \begin{cases} \{v\}, & d = u, \\ \emptyset, & d \neq u. \end{cases}$$

For every choice of the public constants c_1, c_2, c_3, c_4 , every output byte in a column outside E is balanced over the cube. Thus at least $16 - 4|E|$ output bytes are balanced. The resulting 2^{16} -query distinguisher has advantage $1 - 2^{-8(16-4|E|)}$ over a uniform random function.

For each unordered pair $\{u, v\}$, there are 16 choices for the two input rows. Nine choices give $|E| = 0$, six give $|E| = 1$, and one gives $|E| = 2$. Across the six pairs of distinct columns, Proposition 6.1 therefore gives 54 full-output distinguishers, 36 distinguishers on 12 output bytes, and 6 distinguishers on 8 output bytes.

Appendix A proves the proposition and records two independent checks. Exhaustive evaluation over each cube, at the recorded constant and inactive-byte settings, reproduces exactly the 54, 36, and 6 cases above. A bit-based division-property model proves balance for one of the 54 full-output cubes. The same methods produce no four-round certificate for the tested cubes that vary complete input bytes; this marks the reach of the methods, not the absence of other four-round integrals.

6.8 Comparison with a Fixed-Key AES Permutation

Use fixed-key AES [3] as the calibration point. Fix a public AES-128 key k and define

$$P_k(m) := \text{AES}_k(m).$$

The map P_k is a permutation and is intended to resemble a random permutation. In contrast, H_{128} is intended to resemble a random function from 128 bits to 128 bits. The comparison below concerns shared components and reduced-round propagation. It is not a reduction from the security of H_{128} to the security of AES.

Property	Fixed-key AES permutation P_k	Base construction H_{128}
External input	m is the plaintext; the key schedule is fixed.	x determines every round key; the plaintext is fixed to zero.
Target object	Bijjective map with forward and inverse evaluation. Collisions are impossible.	Random-function target. The map need not be bijective, and collisions are part of the target behavior.
Round structure	Standard AES-128 key expansion, nine full rounds, and a final round without MixColumns.	Eleven keys $x + \bar{c}_i$, ten full AESENC-style rounds, and MixColumns in the final round.
Differential input	$\Delta p \neq 0$ and $\Delta K_i = 0$. The reduced-round active-S-box minima are 1, 5, 9.	$\Delta p = 0$ and $\Delta K_i = \Delta x$ in every round. The corresponding minima are 1, 4, 11.
Collision analysis	A same-key collision trail cannot exist because P_k is a permutation.	A relevant trail must additionally have zero output difference. The certified one- through three-round minima are 8, 15, 16.
Integral structure	The classical one-byte Square property gives a three-round distinguisher. Longer AES attacks add partial decryption and key recovery.	Repeated input injection disrupts the one-byte pattern but creates the two-byte family in Proposition 6.1 . There is no secret key to recover.
Linear structure	Every nonzero output-only mask is exactly balanced for uniform plaintext.	Repeated input masks can cancel. The best one-round mask found has output-only correlation of magnitude $2^{-31.256}$. Seven selected complete two-round hulls have magnitude at most $2^{-46.810}$.

Table 7: Qualitative comparison with a fixed-public-key AES permutation. The active-S-box values are certified minima in the recorded truncated models for one, two, and three rounds.

The comparison gives no round-for-round security ordering. Repeated injection of Δx can cancel activity at one boundary and create it at another: the two-round truncated minimum falls from five to four, whereas the three-round minimum rises from nine to eleven. The different final rounds make even these like-for-like model values only a calibration.

More fundamentally, the target objects differ. Collision freedom is correct for P_k , whereas collisions near the birthday bound are correct for a 128-bit random function. The relevant question for H_{128} is whether its input-tied keys yield substantially earlier collisions or correlations. Repeated input injection changes the reduced-round structure, making some measures weaker and others stronger than for fixed-key AES. None of these analyses supplies a full-round bound.

6.9 Extension-Specific Analysis

The two additional constructions introduce cross-tweak relations and a two-lane state. We analyze those mechanisms separately. Any base-recurrence result proved for every public constant schedule also holds for each fixed-tweak slice.

6.9.1 Tweakable Construction

Fix an epoch. The ideal comparison is one random function on the paired domain, or equivalently 256 independent random-function slices indexed by the public tweak. For a fixed tweak, the additive tweak values become part of the public constant schedule and cancel from ordinary differences. Across two tweaks, Equation (1) shows that one message relation can cancel at most one of the eleven complete boundary injections.

The clearest joint data-tweak relation occurs after two rounds. Let $H_{128,a}^{\text{tw},[r]}$ denote the r -round prefix, including its final boundary injection.

Proposition 6.2. *For every epoch a and base message m_0 ,*

$$\bigoplus_{i=0}^{255} H_{128,a}^{\text{tw},[2]}(m_0 + t_1(i), i) = 0^{128}.$$

For 256 independent random functions queried once each, the same equality holds with probability 2^{-128} .

The proof appears in Appendix A. It exploits the linearity of the tweak schedule: choosing $m = m_0 + t_1(i)$ cancels the first tweak injection, leaving algebraic degree below eight before the cube sum. The tested three- and four-round extensions do not retain this balance.

The cross-tweak truncated model certifies minima of 4, 7, and 10 active S-boxes through three rounds for every nonzero tweak difference. An exact local DDT check raises the one-round minimum to six. Differential-linear switching, anchored MITM, and schedule-symmetry screens produce no stronger reduced-round distinguisher. Appendix A also records a schedule-sensitivity check for selected fixed-slice hulls under auxiliary schedules.

6.9.2 Wide Construction

For $j \in [9]$, let $W_j(X)$ be the complete two-lane state after layer j , including the column permutation mix_{256} after even layers. If distinct inputs X, X' satisfy $W_j(X) = W_j(X')$, the next input-fed layer has difference

$$L_{j+1}(X + X'),$$

where L_{j+1} is either the identity or mix_{256} . Because both maps are invertible, the difference is nonzero. Thus an internal full-state collision cannot persist through the next layer, although later nonlinear layers may make the paths reconverge.

For the full 256-bit output, a generic collision costs about 2^{128} . The lowest-weight differential route found forces a full-state collision after layer eight, is separated by the next input injection, and asks the final layer to reconverge. A direct fiber search finds an AES-DDT-compatible four-layer suffix with 86 active S-boxes and nominal weight 602, but does not realize its common internal values or connect it to the six-layer prefix. It is therefore a concrete rebound target, not an attack.

Published preimage results distinguish the two Haraka v2 widths. Attacks on Haraka-256 v2 reach 4.5 of five rounds, whereas the classical attack on Haraka-512 v2 reaches all five rounds [47]. At the time of writing, we are not aware of a published shortcut preimage or partial-preimage attack on all five rounds of Haraka-256 v2.

Schrottenloher and Stevens attack all five rounds of Haraka-512 v2 in a partial-preimage setting [93]. That construction has four AES lanes and a 512-bit permutation state. H_{256} has two lanes and a 256-bit state, so the result is not a direct attack on H_{256} . It nevertheless matters to the random-function target. Matching a prescribed value on 64 fixed output bits of a uniform random function takes 2^{64} evaluations on average. The Haraka attack already accounts for terminal feed-forward. Thus terminal feed-forward alone does not explain why a direct transfer fails.

The relevant structural difference is the repeated input X . The Haraka attack chooses internal values, forms forward and backward lists, and recovers the input by inverting a fixed permutation. In H_{256} , the same unknown X determines the initial state and enters every layer. Backward evaluation therefore depends on X and must recover $X + (\text{iv}_0 \parallel \text{iv}_1)$. The published attack does not impose this consistency condition.

We evaluated one direct two-lane projection of the published internal structure. In the affine knowledge model, the two branches share no additional affine relation beyond their common base relations. A coordinate-guess screen then evaluated 7,560 selected partitions across four choices of eight output-byte positions, each fixed to zero. Every partition used 24 common coordinate guesses and four guesses per branch. In every case, the combined guesses closed the model. All recorded affine, joint-dependency, and shared-S-box comparison metrics were zero.

A separate relaxed byte-color screen represents any eight output-byte positions as fixed. At all nine cuts, the solver returned UNSAT for the predeclared high-dimension condition. An optimistic screening expression motivated that condition; the result does not characterize all attacks below 2^{64} .

All three checks are bounded model results. The partition search does not exhaust closing schedules, local coordinate groups, or choices of output positions. The checks do not cover linear-form guesses, nonlinear joins, or superposition states.

These results do not establish that repeated input makes H_{256} more resistant to partial preimages, and they give no attack on H_{256} . Repeated-input consistency may obstruct the published technique, or a subgeneric adaptation may lie outside our models. The second outcome would violate the intended

random-function profile for partial targets, but would not by itself yield a full collision. The general internal-start class remains open.

The complete-output dependency screen and the original Bao et al. color adaptation exclude the tested anchored two-list routes at every internal cut. Slide checks find no aligned lane or column symmetry. Small-state tests find no predeclared deviation in image size, collision count, preimage multiplicity, or fixed points. Details appear in [Appendix A](#).

Truncating the final output to 128 bits changes the generic collision bound to 2^{64} : outputs with equal first lanes collide even if their second lanes differ. Truncation remains an output option, not a separate construction.

Construction	Evidence reported here	Principal open class
H_{128}	2^{16} -query three-round integral; certified truncated minima 1, 4, 11 and collision-compatible minima 8, 15, 16 through three rounds; selected linear hulls; five-round MITM candidate at 2^{120} ; anchored MITM exclusion for rounds 6–10.	Certified collision bounds beyond three rounds; maximum linear hulls beyond the selected two-round masks; guessed-state, superposition-state, and collision-oriented MITM; and a full ten-round bound.
H_{128}^{tw}	One-boundary cancellation property; 256-query two-round integral; truncated minima 4, 7, 10 through three rounds for all 255 nonzero tweak differences; exact one-round DDT minimum six; auxiliary-schedule hull checks; differential-linear and schedule-symmetry screens; and the anchored MITM model.	Exact two- and three-round differential frontiers; a certified four-round minimum; maximum cross-tweak hulls; guessed-state MITM; and analysis over many epochs.
H_{256}	Internal-collision separation; an 86-active-S-box, weight-602 rebound target after layer eight; complete-output anchored screens and a byte-color screen for eight fixed output bytes at all nine cuts; an affine-knowledge check of one two-lane projection of the Haraka attack; 7,560 selected coordinate-guess partitions across four choices of output positions; 32 tested slide maps; and 256 small-state trials of all 2^{16} inputs.	Common-value and inbound realization of the rebound target; pair-input collisions below 2^{128} ; internal-start partial-preimage attacks below the generic cost for their prescribed output bits; and guessed-state or superposition-state fixed-target preimages below 2^{256} .

Table 8: Quantitative coverage by construction. Propositions are exact statements; solver-certified minima are bounds in their stated models; bounded searches are not security bounds.

The construction-level claims appear in [Table 1](#).

6.10 Model Scope and Reproducibility

The propositions are exact statements about their specified reduced-round functions. Solver-certified minima and exclusions are exact only within their stated truncated, division-property, or anchored MITM models. Complete hull calculations determine correlations for the selected masks. Cube evaluations and small-state checks concern only their recorded inputs and settings; the best solutions found within a time limit do not certify optimality. In particular, the section neither proves ten-round random-function security nor gives a ten-round attack.

Within that scope, the analyses address the main mechanisms suggested by the input-tied recurrence: differential activity, collision compatibility, integrals, linear cancellation, MITM decompositions, cross-tweak relations, and wide-state slides and rebounds. The accompanying artifact contains the models, certificates, exact enumerations, contraction code, and run metadata [1]. [Appendix A](#) gives detailed validation of the reported results.

7 Supporting Ideal-Permutation Analysis

The concrete analysis studies the AES round together with its wrapper. We now remove the round function’s concrete structure and ask a narrower question: does repeated input injection itself create a generic attack when every call uses one public ideal permutation? This analysis guides the wrapper design; it does not select the concrete round count.

Let P be a uniform permutation on $\{0, 1\}^\kappa$ with forward and inverse interfaces. Fix a number $d \geq 1$ of primitive calls and public constants $\mathbf{c} = (c_1, \dots, c_{d+1})$. Define the affine cascade $H_d^{P, \mathbf{c}} : \{0, 1\}^\kappa \rightarrow \{0, 1\}^\kappa$ by

$$q_1 := x + c_1, \quad r_j := P(q_j), \quad q_{j+1} := r_j + x + c_{j+1} \quad (j \in [d]),$$

and output q_{d+1} . This is the H_{128} recurrence with R replaced by P and the depth changed from ten to d .

The target is a uniform function $F : \{0, 1\}^\kappa \rightarrow \{0, 1\}^\kappa$. Write $P^\pm$ for the forward and inverse interfaces. The real interface exposes $(H_d^{P, \mathbf{c}}, P^\pm)$. The ideal interface exposes $(F, S^{F, \mathbf{c}, \pm})$, where the simulator must answer the public-permutation queries. The simulator does not observe the distinguisher’s queries to F [80]. This hidden-query interface is stronger than merely asking for the construction output to look random in isolation.

7.1 Why One and Two Calls Fail

At depth one, public input and output translations reduce the construction to the Matyas–Meyer–Oseas map $u \mapsto P(u) + u$. An inverse query $z := P^{-1}(0)$ forces

$$H_1^{P, \mathbf{c}}(z + c_1) = z + c_1 + c_2.$$

Thus one call cannot realize the target interface.

Depth two retains a forced point. For fixed constants, query

$$z := P^{-1}(c_1 + c_2), \quad x := z + c_1.$$

The two permutation inputs coincide, and the real construction must satisfy

$$H_2^{P,c}(x) = z + c_2 + c_3.$$

Proposition B.1 and **Corollary B.2** give the exact distinguisher.

The forced-point test has advantage at least $1 - (q_S + 1)/2^\kappa$ against a simulator making at most q_S queries to F before answering the inverse query. The test makes its primitive query before any target query, so two calls already fail in the permutation-preprocessing model defined next.

7.2 Three Calls after Permutation Preprocessing

The provable result restricts the query order. We call the access order below the *permutation-preprocessing model*. The distinguisher first makes adaptive forward and inverse queries to P . It then switches to the target interface and queries either the construction or F . It cannot return to the public permutation. Construction evaluations still invoke P internally. The model includes adaptive preprocessing of the public primitive but excludes primitive queries chosen from earlier target outputs.

This access order matches sequential indistinguishability in Cogliati–Seurin and Xu–Da–Guo [29, 103]. Xu–Da–Guo analyze a four-round Even–Mansour cipher with one reused permutation and an affine key schedule. Their target is an ideal cipher over keys and plaintexts. Our target is a one-input random function, so their result and ours do not imply each other.

For depth three, suppose a distinguisher makes at most q_P primitive queries and q_T target queries. Define

$$q := q_P + 3q_T, \quad N := 2^\kappa, \quad B_q := 12(q+1)^2, \quad \Delta_q := N - 4(q+1) - B_q.$$

Theorem 7.1 (Indistinguishability after permutation preprocessing). *Setup samples four public constants independently and uniformly:*

$$c_i \leftarrow \{0, 1\}^\kappa \quad (i \in [4]).$$

There is one simulator $S^{F,c}$ such that every permutation-preprocessing distinguisher D with $q \geq 1$ and $\Delta_q > 0$ satisfies the following bound. The first probability is over setup, P , and the distinguisher's coins. The second is over setup, F , the simulator, and the distinguisher's coins:

$$\left| \Pr[D^{H_3^{P,c}, P^\pm} = 1] - \Pr[D^{F, S^{F,c}, \pm} = 1] \right| \leq \frac{1}{N} + \frac{B_q}{\Delta_q}.$$

The simulator makes at most q queries to F , stores at most $2q$ permutation records, and runs in expected time $O(q)$ with hash-table dictionaries.

Corollary 7.2 (Fixed public schedules). *Fix $c_1, c_2, c_3, c_4 \in \{0, 1\}^\kappa$ with $c_2 \neq c_3$. For every permutation-preprocessing distinguisher D under the query conditions of [Theorem 7.1](#), the same simulator satisfies*

$$\left| \Pr[D^{H_3^{F,c}, P^\pm} = 1] - \Pr[D^{F, S^{F,c}, \pm} = 1] \right| \leq \frac{B_q}{\Delta_q}.$$

The first probability is over P and the distinguisher’s coins. The second is over F , the simulator, and the distinguisher’s coins. The simulator has the same query, storage, and time bounds.

For $q^2 \ll N$, the bound is $O(q^2/N)$. In this model, the third call therefore removes the one- and two-call failures against adaptive permutation preprocessing. The proof, local completion lemmas, and rotated-cycle accounting appear in [Appendix B](#). That appendix also records the open multi-instance and fully interleaved cases.

The theorem applies only in the permutation-preprocessing model. It does not give a standard-indifferentiability theorem for the three-call cascade, a monotonicity statement as calls are added, or an ideal-to-concrete reduction for the AES round. The construction-level boundary for the ten-round function appears in [Table 1](#).

8 Applications

8.1 Checkable Punctured Vectors

Punctured trees let one party obtain every leaf except a selected position, without revealing that position to the party holding the full tree. They support silent OT [\[8\]](#) and symmetric-key signatures [\[15, 16, 28, 14\]](#). Our checkable punctured vector (CPV) interface gives one party a uniform vector and the other all but one entry, together with equality tests for the missing entry. This interface is weaker than a punctured PRF, which hides the missing value even from such tests.

The construction expands each seed $x \in \{0, 1\}^\kappa$ into two children:

$$G(x) := (H(x), H(x) \boxplus x),$$

where $H : \{0, 1\}^\kappa \rightarrow \{0, 1\}^\kappa$ is a random oracle and $\boxplus$ denotes addition modulo 2^κ , used here in place of the XOR in the introduction’s G_H . In a GGM tree [\[57\]](#), the sibling seeds along the path to the missing leaf determine every other leaf. The sum trick of Boyle et al. [\[8\]](#) transfers these seeds using one OT per level. For $n = 2^D$ leaves, full expansion costs $n - 1$ hash calls.

This workload fits H_{128} : every expansion hashes one 128-bit seed to one 128-bit value, and many nodes can be evaluated in parallel. H_{128} is implemented with AES instructions and targets this short-input random-function interface. Instantiating H with H_{128} invokes the construction-level claim in [Table 1](#). [Appendix C](#) gives the functionality, protocol, and complete semi-honest proof in the random-oracle and ideal-OT model.

When the missing leaf is uniform and independent of the adversary’s initial view, equality tests give only a guessing attack (Lemma C.4). Such uses can avoid extra leaf hashes, saving about one third of the primitive calls relative to the pseudorandom-leaf Half-Tree [62]. The correlation-specific Half-Tree already uses approximately n calls; Appendix C.5 compares the interfaces and earlier constructions. The saving applies when CPV suffices; it does not provide PPRF-style hiding or establish a wall-clock advantage over complete protocols.

8.2 Oblivious Transfer Extension

The hashing step in semi-honest OT extension supplies a direct application of the same short-input random-function interface. Let $H : \{0, 1\}^\kappa \rightarrow \{0, 1\}^\kappa$ be a random oracle. In the standard correlation [8, 74], the sender holds s_i and Δ . The receiver holds a choice bit b_i and

$$r_i := s_i + b_i \Delta.$$

Here $+$ is bitwise XOR and $b_i \Delta$ is either 0 or Δ . Hashing converts this correlation to random OT: the sender computes

$$m_0 := H(s_i), \quad m_1 := H(s_i + \Delta),$$

and the receiver computes $H(r_i) = m_{b_i}$.

For $\kappa = 128$, using H_{128} invokes the random-function claim in Table 1 on these correlated inputs. Some maliciously secure protocols also bind a row index [59, 88, 60]. H_{128}^{tw} encodes such a counter through its tweak and epoch, but our analysis does not establish the required tweakable correlation robustness. Proved constructions for that target include permutation-based hashing [59, 33] and the multi-user extension with key leakage [60]. The repository gives a longer comparison [1].

8.3 Other Applications

Garbling schemes hash wire keys together with gate or row identifiers; Free-XOR requires security under circular key correlations [70, 30, 104]. Related hashing interfaces occur in private set intersection [85, 83, 89] and oblivious PRFs [90]. The tweakable construction supplies an indexed domain, but interface compatibility does not establish the correlation properties these protocols require.

Some symmetric-key signature proposals [15, 14] use Rijndael with 256-bit blocks. The wide construction offers a 256-bit hashing interface using AES instructions, but it is not Rijndael-256 and our analysis does not establish those signature schemes’ assumptions. Concrete instantiations of these applications require separate security analyses.

9 Conclusion

Repeated input injection gives a family of fixed-length AES-round functions with a direct hardware implementation. The base construction costs close to fixed-key AES-128 on the measured processor. Our analysis establishes a three-round integral distinguisher and a five-round anchored MITM attack candidate, and examines the additional structure introduced by tweaks and two lanes. [Table 1](#) states the scope of these claims.

The source attack targets Haraka-512 v2; our literature review found no analogous full-round preimage or partial-preimage attack on Haraka-256 v2. For H_{256} , our bounded checks neither transfer the attack nor establish increased resistance. A subgeneric internal-start adaptation remains possible and would weaken the intended partial-preimage profile without itself yielding a collision. We therefore present H_{256} as a design proposal with an open partial-preimage question.

The checkable punctured-vector protocol shows how a fixed-domain random oracle can reduce tree-expansion cost when equality testing of the missing leaf is acceptable. Its proof is in the ideal-OT and random-oracle model. The supporting three-call ideal-permutation theorem addresses the affine wrapper when all primitive queries precede the target queries.

A Details of the Concrete Analysis

This appendix gives the encodings, exact evaluations, solver certificates, and search records behind [Section 6](#). Models, selected patterns, solver outputs, certificates, and reproduction instructions accompany the paper [1]. The repository records each run’s solver, timeout, input hash, and status, and separates proved minima from time-limited incumbents.

A.1 Truncated Related-Key Differential Model

Fix a round count r . For each round, the model has sixteen Boolean variables for the byte activity of the state difference. Sixteen further Boolean variables, shared by all rounds, give the byte activity of Δx . The constraint $\Delta x \neq 0$ requires at least one active key byte. Because the plaintext difference is zero, the activity at the first SubBytes input equals the activity of Δx .

Each round applies the following symbolic transition. SubBytes preserves byte activity and contributes its number of active inputs to the objective. ShiftRows permutes byte positions. For every nonzero MixColumns transition, the number of active input and output bytes is at least five. AddRoundKey forbids exactly one active byte among its two inputs and its output. The same activity vector Δx is injected after every round, and the MixColumns constraint is present in every round because the construction uses the full AESENC-style round throughout. The objective minimizes the total number of active S-boxes.

This is the standard branch-number relaxation used in the CP and MILP literature [58, 22, 32]. It omits exact byte differences, the AES difference-distribution table, and cross-round value consistency.

In this same-instance model, a threshold instance asks whether a full-round collision-compatible trail with at most B active S-boxes exists. At $B = 25$, a 300-second run found no witness and ended at the time limit with status UNKNOWN. This outcome proves neither satisfiability nor unsatisfiability and therefore gives no 26-active lower bound. The reduced-round values in [Table 6](#) instead use objective minimization; the first three are certified optima, whereas the four-round value is an incumbent found within 300 seconds.

We also encoded an exact byte-level model with 8-bit differences, exact XOR, the AES MixColumns map, and difference-distribution-table weights. A 60-second run of its one-round weight optimizer timed out without a witness, so we draw no claim from this model.

A.2 Cross-Instance Truncated Trails

[Table 9](#) gives the complete results under the condition that every byte of every round-key difference is active.

Rounds	Unrestricted trails	Collision-compatible trails
1	16	16
2	16	20
3	20	23
4	23	27 best found

Table 9: Cross-instance truncated trails in the all-active round-key-difference model. All unrestricted minima are certified. The one-round collision minimum is analytical and has a witness. The solver certified the two- and three-round collision minima; the four-round entry is an incumbent found within 60 seconds.

For ten rounds, the cross-instance collision-compatible threshold instance with at most 25 active S-boxes is unsatisfiable in this model. This certificate does not compare directly with the same-instance threshold instance above, which timed out.

A.3 Linear-Mask Search and Hull Contraction

The activity model uses branch number five for the transpose of the AES linear layer. At every input-byte position, cancellation of the repeated external input permits either zero or at least two active masks, but not exactly one. This is only a necessary condition at the activity level. The exact SAT model instead represents byte masks, includes all 60,946 input-output mask pairs with nonzero AES LAT correlation, and applies the bit-level transpose of ShiftRows and MixColumns. A reduced ordered binary decision diagram encodes the LAT support. Adjacent unsatisfiable and satisfiable bounds certify the minima in Table 10.

Write $\lambda_{i,j}$ and $\nu_{i,j}$ for the input and output masks of byte j in round i . We use the signed Walsh-sum convention

$$\text{LAT}(a, b) := \sum_{z \in \{0,1\}^8} (-1)^{\langle a, z \rangle + \langle b, S(z) \rangle}.$$

For a trail θ , its correlation weight is

$$w(\theta) := - \sum_{i,j} \log_2 \left(\frac{|\text{LAT}(\lambda_{i,j}, \nu_{i,j})|}{256} \right),$$

where the sum ranges over the trail's rounds and sixteen byte positions. The trail correlation has magnitude $2^{-w(\theta)}$. Since the maximum nontrivial AES S-box correlation is 2^{-3} , a trail with k active S-boxes has weight at least $3k$.

Rounds	UNSAT bound	Exact minimum	Weight floor	Recorded weight
1	7	8	24	31.256
2	14	15	45	51.746
3	15	16	48	76.098
4	24	25	75	108.296

Table 10: Exact cancellation-compatible linear trails. The minima are certified. The recorded weights are witness values, not optima. The one-round value comes from a time-limited weight search; the two-round value is the strongest trail found while summing one selected hull.

For the two-round hull calculation, the seven selected output masks are

$$\begin{aligned}
\omega_A &= 068300c410061b00\ 0000000001000c17, \\
\omega_B &= 2693007ca3e08200\ 00000000a000c60b, \\
\omega_C &= 2693007ca3e08200\ 00000000a00096fb, \\
\omega_D &= 2693007ca3e08200\ 00000000a000361a, \\
\omega_E &= 970d5a0000000000\ 1b0070a62f2d0014, \\
\omega_F &= 8b0d480000000000\ 1b0070a6262d001d, \\
\omega_G &= 000000002e001e7e\ 4280008203804200.
\end{aligned}$$

For masks A – D , the second-round LAT fixes seven active S-boxes and the first round has at most nine, so the 15- and 16-active layers form the complete hull. The geometries of E , F , and G permit 15 through 18 active S-boxes, and all four layers are included.

The contraction groups masks by AES column and preserves the signs from the S-box LAT and public constants. The 15-active layers were also summed with exact rational arithmetic. For dense 18-active layers, a streamed trace over sparse maps between 16-bit column interfaces replaces the 20.2 GiB pair matrix that direct elimination would require. It agrees with direct contraction on a smaller three-column instance.

Output-mask discovery used two bounded SAT enumerations. A 120-second pass found 1,619 trails and retained 30 masks. A 1,800-second pass found 47,919 trails and reached a retention cap of 10,000 masks. A single-precision GPU screen completed the 18-active layer for 454 of the leading 500 masks; 46 masks with 84, 126, or 252 compatible activity patterns were deferred. Mask A is the output mask of the certified 15-active SAT witness. Masks B , C , D , and E rank 7, 10, 11, and 1 by best sampled trail weight in the 120-second pass. Mask F has the strongest screened 18-active layer, and G is the strongest newly discovered mask from the 1,800-second pass. Complete double-precision contractions, rather than the screen, produce the values in [Table 11](#).

Mask	Trails	$\log_2 C_\omega $	Sampling exponent	Sign-survival ratio
<i>A</i>	1,439,415,224	-52.957	105.914	6.14×10^{-6}
<i>B</i>	1,386,761,412	-49.568	99.136	8.47×10^{-5}
<i>C</i>	1,409,788,499	-48.151	96.301	2.14×10^{-4}
<i>D</i>	1,431,099,138	-49.541	99.082	8.88×10^{-5}
<i>E</i>	7.8694×10^{13}	-47.175	94.350	2.36×10^{-6}
<i>F</i>	7.8648×10^{13}	-46.810	93.620	3.01×10^{-6}
<i>G</i>	7.8634×10^{13}	-47.141	94.282	2.41×10^{-6}

Table 11: Complete two-round linear hulls for seven selected output masks. The sampling exponent is $-2 \log_2 |C_\omega|$. The final column is the magnitude of the signed sum divided by the sum of trail magnitudes.

The single-precision value for mask *E* has relative error 5.1×10^{-8} against the reference contraction. Exact rational sums of the 15-active layers for *F* and *G* provide a second cross-check. These checks validate the reported selected hulls but do not make the bounded selection exhaustive over output masks.

A.4 Proof and Validation of the Three-Round Integral

Proof of Proposition 6.1. Call a byte-valued function of (A, B) *A*-permuting if fixing *B* makes it a permutation of *A*, and define *B*-permuting symmetrically. SubBytes, multiplication by a nonzero MixColumns coefficient, and addition of a function of the other variable preserve this property.

After the first round function, *A* affects every byte in column *u* as an *A*-permuting function. The first feed-forward addition also adds *A* at *U*. In the next ShiftRows operation, *U* moves to column *u*. Consequently, every other destination column receives exactly one *A*-dependent byte, which is *A*-permuting. Every byte of the second round-function output outside column *u* is therefore *A*-permuting. Symmetrically, every byte outside column *v* is *B*-permuting.

The second feed-forward addition can destroy these properties only at *U* and *V*. At *U*, addition of *A* may destroy the *A*-permuting property, but the byte remains *B*-permuting when $c \neq v$. Thus *U* can be exceptional only when $c = v$. Similarly, *V* can be exceptional only when $d = u$.

For every nonexceptional byte $h(A, B)$ entering the third SubBytes layer, either *h* is *A*-permuting or it is *B*-permuting. In the first case,

$$\bigoplus_{A,B} S(h(A, B)) = \bigoplus_B \bigoplus_{z \in \{0,1\}^8} S(z) = 0^8,$$

and the second case is symmetric. The third ShiftRows operation moves the possible exception at *U* to column *u* and the possible exception at *V* to column *v*. MixColumns is linear, so every output byte in a column outside *E* is balanced. The final addition of *x* and *c*₄ has zero XOR sum over the cube.

Let $m := 16 - 4|E|$. For a uniform random function, the selected *m* output bytes at the 2^{16} distinct inputs are independent and uniform. Their XOR is zero with probability 2^{-8m} , whereas it is always zero for the reduced construction. This gives advantage $1 - 2^{-8m}$. $\square$

The exact evaluation first enumerates every value of one active input byte for all 16 byte positions under 16 settings of the remaining bytes. It uses the specified constants and a full AESENC-style linear layer in every round. Every output byte is balanced after one round; after two rounds, each recorded cube balances at least 12 output bytes. No output bit remains balanced across every recorded setting from rounds three through six. These are exhaustive statements for the named cubes and settings, not universal claims.

At three rounds, a second evaluation exhausts all 120 pairs of active-byte positions. Exactly the 96 pairs entering distinct columns after the first ShiftRows operation retain a balanced output byte under one inactive-byte setting, and all 96 survive 16 additional settings. Of these pairs, 54 balance all 16 output bytes, 36 balance 12, and six balance eight. The counts and output columns agree with [Proposition 6.1](#).

For rounds four, five, and six, the initial scan of all 120 two-byte positions returns six, eight, and five apparent byte candidates. None survives 16 additional inactive-byte settings. The four cubes formed by three distinct row-zero bytes also balance no output byte after four rounds in their recorded settings. The conditional-permutation proof stops here because a fourth-round MixColumns input generally contains multiple permuting functions of the same active variable, and the XOR of two permutations need not be a permutation.

The symbolic check uses the exact algebraic-normal-form support of the AES S-box to construct its division transition table. It models ShiftRows, MixColumns, repeated copies of x , and AddRoundKey. An incremental SAT encoding represents the S-box table by a binary decision diagram. For a chosen input cube and output bit, unsatisfiability of the corresponding unit-output query proves balance; satisfiability only means that this model does not prove it.

For the cube with input byte 0 active, the model certifies all 128 output bits balanced after one round. At two rounds, it finds division trails to bits 1–32 and a grouped unsatisfiability certificate for bits 33–128. For the three-round cube at byte positions 0 and 8, four grouped certificates prove all output bits balanced and independently confirm one full-output case of [Proposition 6.1](#).

For four rounds, cyclic column translation reduces the three-, four-, and five-active-byte cube searches to 2,240, 7,280, and 17,472 joint cube–output orbit representatives, covering 107,968 pairs. The model finds a division trail for every representative and therefore certifies no balanced output byte in these classes. This is a limit of the model, not evidence that each output is unbalanced; the symmetry belongs to the symbolic model, not necessarily to the concrete constant schedule.

Rounds	Strongest integral result	Data
1	Every tested one-byte cube balances every output byte	2^8
2	Every tested one-byte cube balances at least 12 output bytes	2^8
3	All 96 cross-column pairs balance 8–16 output bytes	2^{16}
4	No exact two-byte survivor; no 3–5-byte model certificate	$2^{16}_{-2^{40}}$
5–6	No two-byte candidate survives additional-setting tests	2^{16}
7–9	Not evaluated by the exact integral suite	—
10	No concrete distinguisher claimed	—

Table 12: Progression of analytic, exact, and symbolic integral results.

A.5 Public-Tweak Validation

Proof of Proposition 6.2. Set $m_i = m_0 + t_1(i)$. The first primitive input is then $q_1 = m_0 + b_1^{(a)}$, independent of i , and the next state is affine in the eight bits of i . Each coordinate of the second AES S-box layer has degree at most seven in those bits. ShiftRows, MixColumns, and the final affine injection do not increase algebraic degree. The XOR of a Boolean function of degree below eight over the complete eight-dimensional cube is zero, so the equality holds coordinatewise. For a uniform random function on the paired domain, the 256 answers are independent and uniform; their XOR is zero with probability 2^{-128} . $\square$

For cross-tweak differences, we exhaust all 255 nonzero tweak differences and optimize over one common message difference. The minimum aggregate boundary activity is 153 bytes. The truncated model certifies 4, 7, and 10 active S-boxes through three rounds for every nonzero tweak difference. Its four-round value 15 is an incumbent, not a certified minimum. An exact local AES-DDT check gives a one-round minimum of six for every tweak difference; the exact two- and three-round frontiers remain open.

Linear tweak schedules can create joint data–tweak hulls that a fixed-tweak count omits [48]. To measure sensitivity to the public constants, we generated two auxiliary schedules deterministically from seeds 0 and 1. For each schedule, we computed complete two-round contractions for one predeclared output mask at tweaks 0, 31, 32, 255. The eight magnitudes range from $2^{-47.709}$ to $2^{-53.734}$, a spread of 6.025 bits in the exponent. Additive constants change individual trail signs but not the activity structure or trail count. The observed variation therefore comes from cancellation within the hull. The accompanying certificates record the exact schedule derivation and schedule hashes. This check does not evaluate the published AES-counter stream. It also does not bound the maximum over masks or schedules.

The integral evaluator also classifies 96 predeclared 16-dimensional message–tweak cubes. Seventy-six balance all output bits through two rounds, and twenty balance 96 bits. No output bit persists through rounds three or four across 16 predeclared bases in each of epochs 0 and 1 of the published AES-counter stream. We also extend the three-round two-byte integral by varying all 256 tweaks, shifting each message by $t_1(i)$. The resulting 2^{24} -query cube has nonzero XOR after round four.

We also use the AES differential-linear connectivity table [49]. Consider differential-linear routes in which a differential prefix makes the states of two tweaks equal for two consecutive rounds. An attacker-favorable truncated model excludes every such route with data below 2^{64} . A rank-aware adaptation of the Bao et al. color rules [47] is unsatisfiable at every internal cut for both fixed and varying tweaks. The adaptation is attacker-favorable: it grants full-byte control for partial tweak bytes and charges no degrees of freedom for color constraints. It does not cover guessed states, superposition states, nonlinear initial structures outside those rules, or collisions.

Epochs zero and one contain 5,632 effective constants in total, 2,816 per epoch. Exhaustive checks find no equality among the effective constants of an epoch, no two base constants of an epoch whose difference has four equal columns, and no direct slide under the identity coordinate map. Replacing the linear tweak map $i \mapsto t_1(i)$ by evaluated nonlinear maps removes Proposition 6.2 but lowers the minimum aggregate boundary activity from 153 to 142 bytes. The specified linear map has the larger recorded activity and the more complete analysis.

A.6 Wide-Construction Validation

For a fixed difference trail, a base state specifies one member of a pair; the trail’s difference determines the other. A valid base realizes the prescribed AES S-box differences. For the full-state boundary-8 route, the layer-7 and layer-8 base domains have 134 bits of freedom in total. The valid layer-10 bases form a set of size 2^{32} in a 256-bit space. Treating these constraints as independent gives the dimension-count heuristic $256 - 32 - 134 = 90$ bits of deficit. A 120-second exact query times out without deciding the common-value equations. The query does not include the six-layer prefix, so the recorded 86-active, weight-602 suffix is not an attack.

A complete-output dependency screen finds no direct two-list byte match at any cut. A source-ranked adaptation of the Bao et al. color rules is likewise unsatisfiable at all nine cuts. It fixes the 256-bit target, ties each of the 32 input bytes across all occurrences, and charges the degrees of freedom consumed by color constraints. This is a fixed-target preimage screen, not a collision model; it excludes only the anchored two-list route without guessed states or superposition structures.

Schrottenloher and Stevens give a partial-preimage attack on full five-round Haraka-512 v2 [93]. Their algorithm uses internal degrees of freedom and recovers the input by inverting Haraka’s fixed permutation. It also satisfies Haraka’s terminal feed-forward. We reproduced the published witness, including its 64 prescribed zero bits, but did not repeat the 2^{32} search.

Let

$$\text{IV} := \text{iv}_0 \parallel \text{iv}_1.$$

For a fixed input X and boundary t , let $F_{X,t}$ apply the first t layers of H_{256} to an arbitrary 256-bit state. The map $F_{X,t}$ is a permutation. A boundary value

M belongs to the evaluation of H_{256} on X exactly when

$$F_{X,t}^{-1}(M) = X + IV.$$

This equation is the repeated-input consistency condition. The inverse map depends on X , and its result must agree with the initialization defined by the same X . We do not treat the equation as 256 independent random conditions.

We performed three bounded checks involving targets on eight output bytes. The affine checks below fix their selected target bytes to zero. First, we projected the published internal structure onto the two lanes of H_{256} . The projection imposes 24 common byte relations, four forward relations, and four backward byte guesses. After affine propagation and evaluations of S-boxes with one known port, the affine relaxation has nullity 292. It determines 48 of the 672 modeled byte coordinates. The affine knowledge spaces of the two branches have no intersection beyond their common base relations. This calculation does not solve nonlinear relations whose two S-box ports remain unknown.

Second, we tested partitions of 32-coordinate schedules that close the model. Each partition used 24 common guesses and four guesses per branch. A partition passed the structural screen only if:

1. all 32 guesses had the required fresh $24 + 4 + 4$ shape;
2. adding both branches closed the tied-input circuit in both evaluation orders;
3. at least one evaluation order supplied four bytes of joint dependency; and
4. after shared S-box outputs were counted once, the affine intersection beyond the common guesses had rank at least four.

The primary run fixed output bytes zero through seven to zero. It tested 5,040 partitions. A companion run tested 2,520 partitions across three additional choices of eight output-byte positions, again fixed to zero. Every partition had the required fresh shape, and its combined guesses closed the model.

In all 7,560 partitions, the following quantities were zero: the affine intersection rank beyond the common base, the rank after collapsing shared S-box outputs, both order-specific joint-dependency values, and the number of shared S-box relations. Consequently, no tested partition passed the structural screen.

Third, an attacker-favorable byte-color model lets the solver choose which eight output-byte positions are fixed. It treats the other 24 output bytes as backward freedom. Let v be the minimum of the residual forward dimension, the total backward dimension, and the matching dimension, measured in bytes. For screening, we use the optimistic expression

$$2^{8(32-v)}.$$

The condition $v \geq 25$ makes this expression smaller than 2^{64} . The solver returns UNSAT for that condition at all nine cuts. This expression is not an achieved attack runtime.

At threshold one, the model is satisfiable at cuts seven through nine. In each witness, the backward branch obtains all 24 bytes of its nominal freedom from the unfixed output bytes and none from the input. These statuses are recorded solver outputs, not independently checked UNSAT certificates.

None of these checks supplies an attack or a lower bound. The 7,560-partition search samples closing schedules, local coordinate groups, and choices of output positions; it does not exhaust any of these sets. The checks also omit linear-form guesses, arbitrary bit masks, nonlinear joins, superposition states, larger lists, and different merging algorithms. Internal-start partial-preimage attacks therefore remain open.

The slide audit covers the 32 maps formed by independent cyclic column rotations and a lane swap. Only the identity commutes with mix_{256} , and no tested nonzero shift aligns the constant schedule. There is one input for which both lanes repeat their first primitive input in the second layer, but the repetition stops there and yields no known output relation. We also evaluate a toy version with 8-bit lanes in 256 trials. Each trial uses one random 8-bit permutation in both lanes and all ten layers, and exhausts all 2^{16} inputs. The trials show no predeclared deviation in image size, collision count, maximum preimage multiplicity, or fixed points.

Truncating the output to one lane permits a terminal difference of the form $(0^{128}, \Delta_1)$ for any $\Delta_1 \in \{0, 1\}^{128}$. A separate direct fiber search finds an AES-DDT-compatible four-layer suffix with the same 86 active S-boxes and nominal weight 602 under this projected terminal condition. It has the same 90-bit generic dimension deficit and does not include the six-layer prefix. This records the effect of optional output truncation without defining another family member.

A.7 Anchored MITM Nonlinear Checks

The nonlinear checks evaluate the bucket maps used by the anchored MITM candidates. We first define the five-round map from [Figure 2](#). Throughout this subsection, $s_j = q_{j+1}$ is the state after j rounds, $\bar{c}_j = c_{j+1}$, and byte indices use AES column-major order. All byte arithmetic is in the AES field.

Five-round bucket map. Fix the target $Y = s_5$ and the nine gray bytes in $\mathcal{G}$. Let g be the 16-byte vector containing those gray bytes and zero elsewhere. Write

$$A = (\alpha_0, \alpha_1, \alpha_2) := (x[0], x[1], x[2]), \quad E' = (\epsilon_0, \epsilon_1) := (x[5], x[7]).$$

Set $t := Y + \bar{c}_5 + g$. The backward computation first obtains $s_4 = R^{-1}(Y + x + \bar{c}_5)$. To isolate its blue-dependent bytes, define the inverse MixColumns matrix and a four-byte map

$$M := \begin{pmatrix} 0\text{e} & 0\text{b} & 0\text{d} & 0\text{9} \\ 0\text{9} & 0\text{e} & 0\text{b} & 0\text{d} \\ 0\text{d} & 0\text{9} & 0\text{e} & 0\text{b} \\ 0\text{b} & 0\text{d} & 0\text{9} & 0\text{e} \end{pmatrix}, \quad \mathcal{I}(v) := S^{-1}(Mv),$$

where S^{-1} acts coordinatewise on column vectors. Define

$$\begin{aligned}(v_0, v_5, v_{10}, v_{15})^\top &:= \mathcal{I}((t[0] + \alpha_0, t[1] + \alpha_1, t[2] + \alpha_2, t[3])^\top), \\ (v_4, v_9, v_{14}, v_3)^\top &:= \mathcal{I}((t[4], t[5] + \epsilon_0, t[6], t[7] + \epsilon_1)^\top).\end{aligned}$$

The subscripts include the inverse ShiftRows permutation, so $v_j = s_4[j]$ at these eight positions. The first four values depend only on A ; the second four depend only on E' . Set

$$\begin{aligned}\Phi_5(A) &:= \begin{pmatrix} 0\mathbf{b}(v_0 + \alpha_0) + 0\mathbf{d}\alpha_1 + 0\mathbf{9}\alpha_2 \\ 0\mathbf{9}v_5 \\ 0\mathbf{b}v_{10} \\ 0\mathbf{9}v_{15} \end{pmatrix}, \\ \Gamma_5(E') &:= \begin{pmatrix} 0\mathbf{e}v_3 \\ 0\mathbf{d}v_4 + 0\mathbf{9}\epsilon_0 + 0\mathbf{b}\epsilon_1 \\ 0\mathbf{e}v_9 \\ 0\mathbf{d}v_{14} \end{pmatrix}, \\ \eta_B(A, E') &:= \Phi_5(A) + \Gamma_5(E').\end{aligned}$$

Thus $\Phi_5 : \{0, 1\}^{24} \rightarrow \{0, 1\}^{32}$ and $\Gamma_5 : \{0, 1\}^{16} \rightarrow \{0, 1\}^{32}$. Their dependence on Y and the fixed gray assignment is suppressed in the notation.

To see why this is the bucket key, form $u_3 := \mathbf{MC}^{-1}(s_4 + x + \bar{c}_4)$. Expanding rows of M shows that

$$(u_3[3], u_3[6], u_3[9], u_3[12])^\top = \eta_B(A, E') + \rho,$$

where ρ depends on the red bytes, gray bytes, target, and constants, but not on A or E' . The displayed maps contain every blue-dependent term in these four coordinates. Assignments in one bucket therefore give the same four values for each fixed red assignment. Inverse ShiftRows and SubBytes recover the whole last column of s_3 . That column of x contains no blue bytes. The next inverse round therefore recovers the backward-known cut bytes $\{1, 6, 11, 12\}$, including the match byte $s_2[6]$, independently of the blue representative in the bucket.

For a bucket key $\eta \in \{0, 1\}^{32}$, define its fiber

$$\mathcal{B}_\eta := \{(A, E') : \Phi_5(A) + \Gamma_5(E') = \eta\}.$$

Averaged over all 2^{32} keys, including empty fibers, its size is $2^{40}/2^{32} = 2^8$. This average does not require injectivity of either component map. For the all-zero gray assignment, the artifact exhausts both component domains at targets

$$Y_0 = 0^{128}, \quad Y_1 = 54\mathbf{b}90\mathbf{b}3\mathbf{c}3\mathbf{a}\mathbf{e}\mathbf{d}7\mathbf{b}\mathbf{d}212051\mathbf{c}69\mathbf{d}4\mathbf{e}52\mathbf{e}\mathbf{b}0.$$

Both component maps are injective at each target. The bucket containing the all-zero blue assignment has size 239 at Y_0 and 256 at Y_1 . A bounded search over further keys finds fibers of sizes 285 and 269, respectively; these are the largest recorded sizes, not certified maxima.

Four-round bucket map. For the four-round check, fix $Y = s_4$. The blue (forward-neutral) bytes are $\{4, 5, 6, 7, 13, 14\}$; set every other byte of x to zero. Parameterize the blue bytes by $A = (\alpha_0, \alpha_1, \alpha_2)$ and one byte ϵ :

$$\begin{aligned} (x[4], x[5], x[6], x[7], x[13], x[14]) &= (\alpha_0, \alpha_1, \alpha_2, \delta, \epsilon, \varphi), \\ \delta &:= 0\mathbf{e}^{-1}(0\mathbf{b}\alpha_0 + 0\mathbf{d}\alpha_1 + 0\mathbf{9}\alpha_2), \\ \varphi &:= 0\mathbf{b}^{-1}0\mathbf{e}\epsilon. \end{aligned}$$

These two linear equations keep coordinates 7 and 13 of $\mathbf{MC}^{-1}(Y + x + \bar{c}_4)$ fixed. Let π_J select the coordinates in the ordered tuple $J = (5, 8, 15)$, and define

$$\Psi_4(A, \epsilon) := \pi_J(\mathbf{MC}^{-1}(R^{-1}(Y + x + \bar{c}_4) + x + \bar{c}_3)).$$

The variables A and ϵ enter different columns before the last inverse Mix-Columns operation. No inverse S-box in this expression depends on both. Consequently,

$$\begin{aligned} \Phi_4(A) &:= \Psi_4(A, 0), & \Gamma_4(\epsilon) &:= \Psi_4(0, \epsilon) + \Psi_4(0, 0), \\ \Psi_4(A, \epsilon) &= \Phi_4(A) + \Gamma_4(\epsilon). \end{aligned}$$

Here Φ_4 maps three bytes to three bytes, and Γ_4 maps one byte to three bytes. At each of the same two targets Y_0, Y_1 , exhaustive evaluation proves that Φ_4 is a permutation. Every three-byte bucket key therefore has exactly 2^8 preimages under Ψ_4 .

B Detailed Ideal-Permutation Analysis

This appendix gives the attacks, auxiliary lemmas, and proofs summarized in [Section 7](#). It also records the unresolved extensions.

We use an ideal permutation for a controlled structural analysis. The analysis preserves the affine wrapper and reuse of one public permutation, but removes weaknesses specific to the AES round function. A generic attack in this setting would therefore expose the wrapper itself. Conversely, an ideal-permutation theorem would not establish security after replacing the permutation by a concrete AES round.

Fix a number $d \geq 1$ of primitive calls, a permutation P on $\{0, 1\}^\kappa$, and a public schedule $\mathbf{c} = (c_1, \dots, c_{d+1}) \in (\{0, 1\}^\kappa)^{d+1}$. Write $N := 2^\kappa$. The d -call *affine cascade* $H_d^{P, \mathbf{c}} : \{0, 1\}^\kappa \rightarrow \{0, 1\}^\kappa$ sets

$$q_1 := x + c_1, \quad r_j := P(q_j), \quad q_{j+1} := r_j + x + c_{j+1} \quad (j \in [d]),$$

and outputs q_{d+1} . All additions in this section are XOR. The unrestricted target notion is standard indistinguishability, where the distinguisher may interleave target queries with forward and inverse queries to the reused permutation. One and two calls fail even in the permutation-preprocessing model of [Section 7](#). For three calls, we prove indistinguishability in that model.

B.1 The one-call baseline

At depth one, public input and output translations reduce the cascade to the Matyas–Meyer–Oseas map $H_{\text{MMO}}(u) := P(u) + u$:

$$H_1^{P,c}(x) = H_{\text{MMO}}(x + c_1) + c_1 + c_2.$$

This map has useful restricted correlation-robustness properties [59, 44, 33], but it does not realize the target interface when the permutation can be queried in both directions [9, 10, 11]. Querying $P^{-1}(0)$ returns a value z for which the real map must satisfy $H_{\text{MMO}}(z) = z$. For the translated cascade, the test is $H_1^{P,c}(z + c_1) = z + c_1 + c_2$. A simulator making at most q_S queries to F before returning z finds such a point with probability at most $(q_S + 1)/N$. Each fresh queried input has one prescribed successful output; an unqueried final candidate adds one more trial. The test therefore gives advantage at least $1 - (q_S + 1)/N$. The next question is whether a second call removes this obstruction.

B.2 Why two calls fail

Write $H_2 := H_2^{P,c}$. The two-call cascade is given in Figure 3. Setup samples and publishes an independent constant for each state transition, matching the constant placement of the AES-derived construction.

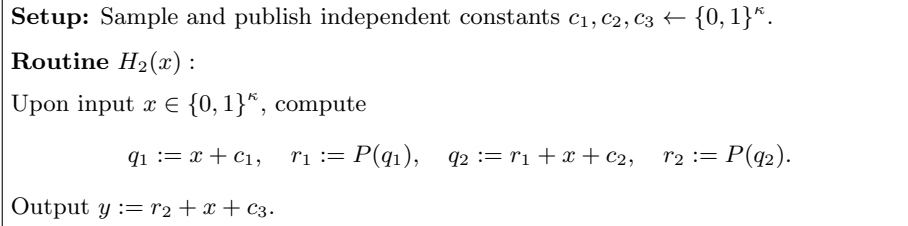


Figure 3: The two-call affine cascade $H_2 : \{0, 1\}^\kappa \rightarrow \{0, 1\}^\kappa$ with the same constant placement as the concrete AES-derived construction.

The natural simulation strategy is eager chain completion. When a simulated permutation query identifies an input x , the simulator queries $F(x)$ and programs an unused permutation entry so that the completed chain outputs $F(x)$. The strategy requires at least one unassigned entry in the chain. The construction has one input for which no such entry remains.

Proposition B.1. *Fix a permutation P and constants c_1, c_2, c_3 . There is exactly one input x^* for which the two permutation inputs in H_2 coincide. Define*

$$t := c_1 + c_2, \quad z^* := P^{-1}(t), \quad x^* := z^* + c_1.$$

Then $q_1 = q_2 = z^*$ and

$$H_2^P(x^*) = z^* + c_2 + c_3.$$

Proof. Write $z := x + c_1$, so $q_1 = z$. The equality $q_1 = q_2$ holds exactly when

$$z = P(z) + (z + c_1) + c_2,$$

or equivalently when $P(z) = c_1 + c_2 = t$. The permutation P has exactly one such input, namely $z^* = P^{-1}(t)$. For $x^* = z^* + c_1$, both permutation calls therefore return t , and

$$H_2^P(x^*) = t + x^* + c_3 = z^* + c_2 + c_3.$$

□

We now trace the eager simulator on the exceptional inverse query. Write $\tilde{P}$ for the permutation simulated by the eager simulator. The query $\tilde{P}^{-1}(t)$ makes the simulator return some z and fixes $\tilde{P}(z) = t$. The simulator recognizes the corresponding construction input $x := z + c_1$ and queries $w := F(x)$. Completing the chain would require

$$\tilde{P}(q_2) := w + x + c_3.$$

However, $q_2 = z$, and the inverse query has already fixed $\tilde{P}(z) = t$. The two assignments are consistent only if

$$F(z + c_1) = z + c_2 + c_3. \quad (2)$$

Thus the exceptional input leaves no free permutation entry with which to program the target-function value.

Corollary B.2. *Fix constants $c_1, c_2, c_3 \in \{0, 1\}^\kappa$ and set $t := c_1 + c_2$. Let S^F be a simulator that makes at most q_S queries to F before answering one inverse-permutation query. There is a distinguisher that makes one inverse-permutation query and one target query with indistinguishability advantage at least*

$$1 - \frac{q_S + 1}{2^\kappa}.$$

Proof. The distinguisher queries $z := P^{-1}(t)$, sets $x := z + c_1$, queries the left oracle at x , and checks Equation (2). Proposition B.1 shows that the check always succeeds in the real world.

In the ideal world, the simulator can return a candidate derived from one of its target-function queries. For each new queried input, its fresh uniform answer satisfies Equation (2) with probability $2^{-\kappa}$, conditioned on the preceding transcript. Repeated queries add no candidates. If the simulator returns an unqueried candidate, the distinguisher's query succeeds with the same probability. A union bound gives ideal-world success probability at most $(q_S + 1)/2^\kappa$. □

The forced-point test already proves two-call impossibility. The accompanying repository gives a separate interleaved endpoint attack and its hidden-path analysis [1].

B.3 Three calls after permutation preprocessing

Write $H_3 := H_3^{P,c}$. The three-call cascade in Figure 4 uses the same affine cascade and the same permutation. A first edge and one construction output no longer determine the middle edge. The third call therefore creates a candidate edge through which a simulator can complete the evaluation. This local freedom is necessary but not sufficient for fully interleaved access. We obtain an indistinguishability theorem after adaptive permutation preprocessing.

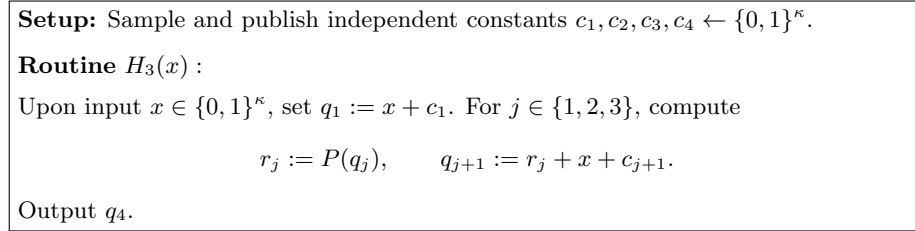


Figure 4: The three-call construction $H_3 : \{0, 1\}^\kappa \rightarrow \{0, 1\}^\kappa$ with independent public round constants.

Section 7 defines the permutation-preprocessing model, a form of sequential oracle access [81, 103], and compares it with the sequential-indistinguishability results of Cogliati–Seurin and Xu–Da–Guo [29, 103]. If the distinguisher makes at most q_P primitive queries and q_T target queries, we charge total permutation-query cost $q := q_P + 3q_T$. This syntactic cost is the same in both games. Theorem 7.1 states the resulting bound. The random setup includes the degenerate schedule $c_2 = c_3$; Corollary 7.2 removes that term for every fixed nondegenerate schedule.

We next isolate the local completion rule, the failure of repeatable chaining, and the rare cycle that the restricted proof must charge. We also record a backward-query example outside the theorem, which identifies work required by the fully interleaved case. We begin with the edge available to an eager simulator and the exceptional public schedule for which that edge also collapses.

Proposition B.3. *Fix a permutation P and constants c_1, c_2, c_3, c_4 . Let*

$$t := c_1 + c_2, \quad z := P^{-1}(t), \quad x := z + c_1.$$

Then the first two permutation inputs of $H_3(x)$ equal z , and its third permutation input is

$$q_3 = z + c_2 + c_3.$$

Consequently:

1. *If $c_2 = c_3$, all three permutation inputs equal z and $H_3(x) = z + c_2 + c_4$.*
2. *If $c_2 \neq c_3$, then $q_3 \neq z$.*

For the second case, consider a partial permutation table containing (z, t) , and fix a desired output $w \in \{0, 1\}^\kappa$. If q_3 is unassigned and $w + x + c_4$ is unused as a table output, inserting

$$P(q_3) := w + x + c_4$$

is consistent with the table and forces $H_3(x) = w$ for every permutation extending the updated table.

Proof. **Proposition B.1** gives $q_1 = q_2 = z$ and makes both calls return t . Substitution gives

$$q_3 = t + x + c_3 = z + c_2 + c_3.$$

Thus $q_3 = z$ exactly when $c_2 = c_3$. In that case the third call also returns t , and the displayed value of $H_3(x)$ follows. Otherwise q_3 is distinct from z , whose entry (z, t) is already in the table. The stated assignment then makes $q_4 = P(q_3) + x + c_4 = w$. $\square$

What fully interleaved access adds. The completion in **Proposition B.3** programs the third edge after the first two are exposed. A simulator must also recognize chains exposed from the output backward or from both ends. The next proposition gives the required algebra. Write $E_j := (q_j, r_j)$ for the j th permutation edge of an H_3 evaluation.

Proposition B.4. *Fix a target output w . Any two edges that are compatible with an evaluation returning w determine its input x and the third edge as follows:*

1. *Given $E_1 = (q_1, r_1)$ and $E_2 = (q_2, r_2)$, define $x := q_1 + c_1$. If $q_2 = r_1 + x + c_2$, the required third edge is*

$$E_3 = (r_2 + x + c_3, w + x + c_4).$$

2. *Given $E_2 = (q_2, r_2)$ and $E_3 = (q_3, r_3)$, define $x := q_3 + r_2 + c_3$. If $r_3 = w + x + c_4$, the required first edge is*

$$E_1 = (x + c_1, q_2 + x + c_2).$$

3. *Given $E_1 = (q_1, r_1)$ and $E_3 = (q_3, r_3)$, define $x := q_1 + c_1$. If $r_3 = w + x + c_4$, the required middle edge is*

$$E_2 = (r_1 + x + c_2, q_3 + x + c_3).$$

In every case, adding the required edge completes an evaluation with output w , provided that its input and output are unused or already assigned to each other.

Proof. Each displayed edge follows by solving the defining equations in **Figure 4**. The condition $r_3 = w + x + c_4$ is necessary in the last two cases because the third edge is already fixed. $\square$

This proposition handles a natural backward-chain test. A distinguisher may query the left oracle at x , receive w , and then compute

$$q_3 := P^{-1}(w + x + c_4), \quad q_2 := P^{-1}(q_3 + x + c_3), \quad q_1 := P^{-1}(q_2 + x + c_2).$$

In the real world, it always obtains $q_1 = x + c_1$. In the ideal world, after answering the first inverse query, a simulator receives the second inverse query at $r_2 = q_3 + x + c_3$. It can recover $x = r_2 + q_3 + c_3$, query $F(x)$, recognize the target-consistent suffix, and apply Item 2 of [Proposition B.4](#) before returning. Thus a simulator that only completes chains from the input side fails this test, whereas bidirectional completion has one edge left to program.

The resulting proof strategy is to scan each new permutation edge against the current table and complete every target-consistent two-edge chain. This is the same high-level issue addressed by chain-completion simulators for three-round Even–Mansour constructions [\[55\]](#). That result does not prove security here: it uses independent round permutations and a non-invertible random-function key derivation, while H_3 reuses one permutation and injects x directly.

Long-chain attacks are the main reason to test this strategy carefully. Dodis et al. show that a simulator for a second iterate may have to prepare chains as long as the distinguisher’s query budget [\[39\]](#). Guo and Lin encounter the corresponding pressure for three-round Even–Mansour and use a preemptive rhizome mechanism [\[55\]](#). More recently, Bhaumik et al. give a generic chaining attack on broad classes of two-call permutation constructions [\[20\]](#). Their attack repeatedly turns one construction answer into the next permutation edge and checks only the two endpoints. The next proposition isolates why that particular recurrence does not continue in H_3 .

Proposition B.5 (No repeatable two-edge shift). *Assume $c_2 \neq c_3$. A partial permutation cannot contain three construction chains with respective edge sequences*

$$(E_1, E_2, E_3), \quad (E_2, E_3, E_4), \quad (E_3, E_4, E_5).$$

Proof. Here $E_1, \dots, E_5$ are arbitrary table edges, not the positional edges of one evaluation, and $E.\text{out} := v$ for an edge $E = (u, v)$. Write x and x' for the inputs of the first two chains. Their equations for the input of E_3 give $E_2.\text{out} + x + c_3 = E_2.\text{out} + x' + c_2$. Their equations for the input of E_2 give $E_1.\text{out} + x + c_2 = x' + c_1$. Together these imply

$$x' = x + c_2 + c_3 \quad \text{and} \quad E_1.\text{out} = c_1 + c_3.$$

Applying the same calculation to the second and third chains gives $E_2.\text{out} = c_1 + c_3$. Injectivity of the permutation therefore gives $E_1 = E_2$. Comparing their inputs in the first chain then gives $c_2 = c_3$, a contradiction. $\square$

Thus a sequence of chains cannot repeat this two-edge shift twice; [Proposition B.6](#) analyzes a pair with one such overlap. [Proposition B.5](#) rules out the repeatable-shift mechanism of the generic two-call attack. It is not an indistinguishability proof. In particular, chains that share one edge or meet through

unrelated table entries can still create recursive completion and programming conflicts.

The remaining task is global rather than algebraic. Several recognized chains may request incompatible assignments, and a programmed edge may create new two-edge candidates. Two representative collision surfaces are

$$q \mapsto P(q) + q \quad \text{and} \quad x \mapsto F(x) + x.$$

Collisions in the first map allow one permutation input to serve as the second edge of multiple prefixes; collisions in the second allow multiple target values to request the same permutation output after translation. A proof must bound these collisions, premature table assignments, and the length of the resulting completion cascade under adaptive queries. **Proposition B.4** shows that query direction alone creates no additional local obstruction.

There is, however, a rare two-input cycle in which two chains request only already assigned edges. The cycle gives a concrete lower bound that any proof must accommodate.

Proposition B.6. *Fix a permutation P on $\{0, 1\}^\kappa$ and constants $c_1, c_2, c_3, c_4 \in \{0, 1\}^\kappa$. Assume $N \geq 4$ and $c_2 \neq c_3$, and define*

$$\delta := c_2 + c_3, \quad z := P^{-1}(c_1 + c_3),$$

and set

$$x := z + c_1, \quad x' := x + \delta, \quad \tau := z + c_2 + c_4.$$

The first and second permutation edges of $H_3(x')$ are respectively the second and third edges of $H_3(x)$. Moreover,

$$H_3(x) = \tau \implies H_3(x') = \tau.$$

For a uniformly random permutation P ,

$$\Pr[H_3^P(x) = \tau] = \frac{2}{N-1}.$$

Proof. Write $a := c_1 + c_3$ and $b := P(z + \delta)$. The first three edges of the evaluation at x have inputs and outputs

$$E_1 = (z, a), \quad E_2 = (z + \delta, b), \quad E_3 = (b + z + a, \gamma),$$

where $\gamma := P(b + z + a)$. Since $x' = z + c_1 + \delta$, direct substitution shows that the first two edges at x' are E_2 and E_3 .

The equality $H_3(x) = \tau$ is equivalent to $\gamma = c_1 + c_2$. Under this condition, the third permutation input at x' is z . Its third edge is therefore E_1 , and

$$H_3(x') = P(z) + x' + c_4 = z + c_2 + c_4 = \tau.$$

It remains to calculate the event probability. Condition on $z = P^{-1}(a)$; the remaining permutation is uniform subject to $P(z) = a$. The value $b = P(z + \delta)$

is uniform over the $N - 1$ outputs other than a . Since $c_2 \neq c_3$, the target $c_1 + c_2$ differs from a . If $b = c_1 + c_2$, which occurs with probability $1/(N - 1)$, then $b + z + a = z + \delta$ and the event holds. Otherwise, $b + z + a$ is distinct from z and $z + \delta$, and its permutation output equals the available target $c_1 + c_2$ with probability $1/(N - 2)$. Thus the probability is

$$\frac{1}{N - 1} + \frac{N - 2}{N - 1} \cdot \frac{1}{N - 2} = \frac{2}{N - 1}.$$

The value is the same for every z , so averaging removes the conditioning. $\square$

Corollary B.7. *Fix constants $c_1, c_2, c_3, c_4 \in \{0, 1\}^\kappa$ with $c_2 \neq c_3$, assume $N \geq 4$, and set $\delta := c_2 + c_3$. Let S^F make at most q_S queries to F before answering one inverse-permutation query. There is a distinguisher making one inverse query and two left-oracle queries with advantage at least*

$$\frac{2}{N - 1} - \frac{(q_S + 1)^2}{N^2}.$$

Proof. The distinguisher queries $z := P^{-1}(c_1 + c_3)$, derives x, x', τ as in [Proposition B.6](#), queries the left oracle at x and x' , and outputs one exactly when both answers equal τ . [Proposition B.6](#) gives real-world success probability $2/(N - 1)$.

In the ideal world, every candidate z requires the two distinct target-function conditions

$$F(z + c_1) = F(z + c_1 + \delta) = z + c_2 + c_4.$$

Count only distinct simulator queries. A queried input u can occupy either position in a successful pair. These orientations prescribe the respective targets $u + c_1 + c_2 + c_4$ and $u + \delta + c_1 + c_2 + c_4$ before $F(u)$ is sampled. For either orientation, the other required input is $u + \delta$.

If both pair members are queried, fix their first-query times and the orientation of the earlier member. Its prescribed equality has probability $1/N$. Conditioned on the history before the later member's first query, that member's prescribed equality also has probability $1/N$. There are at most $2 \binom{q_S}{2} = q_S(q_S - 1)$ choices of times and orientation. Their total contribution is at most $q_S(q_S - 1)/N^2$.

If exactly one member is queried, the same first-equality argument has at most $2q_S$ choices. The unqueried partner remains uniform when the simulator returns its candidate, giving a contribution at most $2q_S/N^2$. If neither member is queried, both values remain independent uniform values at that point, giving at most $1/N^2$. These cases bound ideal success by

$$\frac{q_S(q_S - 1) + 2q_S + 1}{N^2} \leq \frac{(q_S + 1)^2}{N^2}.$$

Subtracting from real success proves the claim. $\square$

This cycle is a genuine programming obstruction, but its distinguishing advantage is of order $2^{-\kappa}$ for polynomially bounded q_S . It therefore permits a

conventional collision-style indistinguishability bound. A coupling may charge the cycle as a bad event, but must bound its probability rather than assume it cannot occur.

Proof in the permutation-preprocessing model. For independently sampled constants, the collapsed schedule $c_2 = c_3$ occurs with probability $2^{-\kappa}$. In the permutation-preprocessing model, the simulator needs only forward-prefix completion: it programs the third edge after the first two construction edges become public. Internally programmed edges remain private until a later permutation call exposes them, so insertion does not start recursive closure. The rotated cycle becomes an ordinary collision-order bad event. We now prove [Theorem 7.1](#).

Proof of Theorem 7.1. We describe one simulator that is independent of the distinguisher. If $c_2 = c_3$, it answers primitive queries as an ordinary lazy uniform permutation and makes no queries to F . The analysis charges this schedule as a bad setup event. For the remaining case, fix a public schedule $\mathbf{c}$ with $c_2 \neq c_3$, and write S^F for $S^{F,\mathbf{c}}$. Also fix the random tape of an arbitrary permutation-preprocessing distinguisher D . We first prove the bound for this deterministic distinguisher and fixed nondegenerate schedule, and then average over the coins of D and the setup.

We use three games. Game G_R is the real game with a lazy uniform permutation P . Game G_M samples a lazy uniform function F and uses the augmented permutation procedure defined below. During the target phase, G_M answers a left-oracle query by evaluating H_3 through that procedure. Game G_I uses the same procedure during the primitive phase, but answers target-phase queries directly with F . The simulator S^F in the theorem is the primitive-phase restriction of the augmented procedure. Thus G_I is the ideal game. The private target-phase evolution in G_M is only a proof device; D cannot query the primitive after switching phases.

The procedure below maintains a bad flag. Once the flag is set, it completes the current answer consistently and continues as an ordinary lazy permutation. Its behavior after this point is irrelevant to the coupling.

Prefix completion. Write a permutation edge as $E = (u, v)$ and set $h(E) := u + v$. Edges $E_1 = (u_1, v_1)$ and $E_2 = (u_2, v_2)$ form the first two edges of one H_3 evaluation exactly when

$$u_2 = h(E_1) + c_1 + c_2. \quad (3)$$

For such a pair, set $x := u_1 + c_1$. The required third edge is

$$E_3 := (v_2 + x + c_3, F(x) + x + c_4). \quad (4)$$

For a fixed partial permutation and fixed x , the pair (E_1, E_2) is unique: x fixes u_1 , which fixes v_1 , then u_2 , and then v_2 .

The augmented procedure maintains a partial permutation as disjoint sets Π_{pub} and Π_{in} , both initially empty. The bad flag is initially unset. A record

becomes public after the procedure returns it either to D in the primitive phase or to a construction evaluation in G_M . A record created by Equation (4) is internal until a later permutation call exposes it. A *publication cycle* consists of publishing one record and performing the checks and possible completion below. A *completion* is one attempted insertion of Equation (4). The procedure maintains dictionaries for permutation inputs, permutation outputs, private outputs, and $h(E)$ for public records.

A fresh forward query receives a uniform unused output and is labeled $\rightarrow$. A fresh inverse query receives a uniform unused input and is labeled $\leftarrow$. A forward query at the input of an internal record promotes that record to Π_{pub} and labels the exposure $\perp \rightarrow$. An inverse query at the output of an internal record sets the bad flag and returns the assigned input. Repeated public queries return their recorded answers.

Before publishing any record $E = (u, v)$, the procedure sets the bad flag if u equals the output of an internal record. The input u is the supplied input of a forward query, the input sampled for a fresh inverse query, or the input of a promoted record; in the last case, the compared outputs include the output of E itself. The check prevents a private output from becoming a known permutation input.

Suppose that $E = (u, v)$ has just become public. The procedure performs the following checks before returning its answer.

1. It sets the bad flag if a distinct public edge has the same h value as E .
2. It looks for a public E_1 with $h(E_1) = u + c_1 + c_2$. Unless check 1 sets the flag, there is at most one.
3. If such an E_1 exists, $E_1 \neq E$, and the label of E is $\rightarrow$ or $\perp \rightarrow$, the procedure inserts the edge in Equation (4). It also inserts the completion of the unary inverse prefix $E_1 = E = (u, c_1 + c_2)$, whose requested third input is $u + c_2 + c_3 \neq u$.
4. Every other newly exposed prefix sets the bad flag. In particular, a unary prefix exposed forward or by promotion has tested the fresh equality $v = c_1 + c_2$.
5. It sets the bad flag if an older public edge has input $h(E) + c_1 + c_2$. This is a prefix whose first edge became public second.

When inserting Equation (4), the procedure sets the bad flag if the requested input is already assigned, the requested output is already used, the input equals a private output, or the output equals an assigned input. The last check includes the requested input itself. Thus an internal fixed point also sets the flag. Otherwise the procedure inserts the edge into Π_{in} . An internal insertion does not process further pairs.

Outside an h collision, a newly public record is the second edge of at most one prefix. Hence each public record creates at most one internal record. If at

most q permutation calls occur, then

$$|\Pi_{\text{pub}}| \leq q, \quad |\Pi_{\text{in}}| \leq q, \quad |\Pi_{\text{pub}} \cup \Pi_{\text{in}}| \leq 2q. \quad (5)$$

The augmented procedure, and therefore the simulator, makes at most q queries to F . Each call performs at most one completion and, apart from rejection sampling, $O(1)$ dictionary operations. At most $2q$ inputs and $2q$ outputs are assigned. Since $\Delta_q > 0$ implies $4q < N$, each fresh permutation sample requires fewer than two trials in expectation. With hash-table dictionaries, the simulator therefore runs in expected time $O(q)$.

Privacy invariant. Immediately before every publication cycle, each internal output V has three properties: it has not been returned through the permutation procedure; it does not occur as an assigned permutation input; and, given the exposed information described below, it is uniform over the values not yet excluded. That exposure leaves unexposed only the function value defining V and the state entries determined by that value. We call such an output a private pivot.

We prove the invariant by induction over cycles. An internal record created for input x has output $F(x) + x + c_4$. The function input x is fresh. Two processed prefixes for the same x would be the same pair by prefix uniqueness, and one pair cannot become new twice. If the target phase of G_M had already answered at x , that construction evaluation would have exposed the unique third edge before returning. The insertion checks exclude used outputs and every assigned input, including the new edge's own input.

Before a later permutation call can place a private output in the input table, the prepublication check sets the bad flag. An inverse query at the private output also sets the flag. If a forward query asks for the internal input, the procedure processes the promoted record before returning its output. Consequently the output remains a private pivot until that promotion cycle, and the induction continues for every other internal record.

To justify conditional uniformity, fix a history ending before a private output V is returned. Expose every random sample in this history except the function sample defining V . Retain V as a variable in its record; do not condition on that record's literal output. Every internal input was computed from public prefix coordinates at insertion. Until V is exposed, those coordinates and all subsequent query choices are unchanged when V varies along a history with no bad event. Membership tests and the explicit equality checks can exclude values of V , but reveal no other function of it.

Every remaining value of V gives the fixed history the same probability. The defining function sample has probability $1/N$. Each later function sample also has probability $1/N$, and each fresh permutation sample has probability $1/(N - m)$, where m is the number of assigned records at that step. These record counts are the same for every admissible value of V . Conditioning on later samples therefore only excludes values that would violate table membership or a failed equality test. It does not weight the remaining values differently.

Each such exclusion fixes one value of V after the other samples are exposed. This proves the asserted uniformity, including immediately before a promotion returns V .

One-cycle analysis. Use the exposure from the privacy invariant immediately before a cycle, and condition on the absence of an earlier bad event. For a fresh forward record, the unsupplied output is a fresh pivot. For a fresh inverse record, the unsupplied input is a fresh pivot. A coordinate supplied by D is fixed before either pivot is sampled. A private output is a pivot by the preceding invariant.

An h collision fixes the new record's pivot. A unary prefix exposed forward or by promotion fixes its output pivot to $c_1 + c_2$. If a distinct inverse record serves as E_2 , Equation (3) fixes its input pivot. If the new record serves as E_1 of an older E_2 , the same equation fixes the new record's pivot.

An inverse request at a private output fixes that private pivot. If the input of a newly public forward record equals a private output, the supplied input fixes the private pivot. If an input sampled for a fresh inverse record equals a private output, we instead condition on the private outputs and charge the fresh inverse-input pivot.

If a distinct forward or promoted record serves as E_2 , the requested input in Equation (4) contains its output pivot exactly once. A preassigned input or a hit on a private output therefore fixes that pivot. The requested output contains the independent fresh value $F(x)$ exactly once. Hence a used-output or assigned-input hit, including equality with the requested input, fixes the function-output pivot. For the unary inverse prefix, the input pivot u occurs once in the requested third input $u + c_2 + c_3$. This value differs from u because $c_2 \neq c_3$, and the requested output contains the fresh value $F(u + c_1)$. Thus every bad event fixes one value of a conditionally uniform pivot, and every successful insertion preserves the privacy invariant.

Probability bound. Every publication cycle is triggered by a charged permutation call, so there are at most q cycles. Prefix uniqueness gives at most one completion per cycle and hence at most q completions.

The following families cover the branches that set the bad flag. The last column bounds the candidate equalities tested over the complete execution. Here “assigned” ranges over $\Pi_{\text{pub}} \cup \Pi_{\text{in}}$, which contains at most $2q$ records, and “private” ranges over at most q internal outputs.

Family	Flag-setting equality	Candidates
B_h	$h(E)$ equals that of an earlier public edge	$q(q-1)/2$
B_{un}	a forward or promoted unary prefix has $v = c_1 + c_2$	q
$B_{\leftarrow 2}$	a distinct inverse-labeled record is the second edge of a prefix	q^2
B_{rev}	a newly public record is the first edge of an older prefix	q^2
$B_{\leftarrow \text{priv}}$	an inverse query names a private output	q^2
$B_{\text{in-priv}}$	the input of a published record equals a private output	q^2
$B_{3,\text{in}}$	the requested third input is assigned	$2q^2$
$B_{3,\text{priv}}$	the requested third input equals a private output	q^2
$B_{3,\text{out}}$	the requested third output is used	$2q^2$
$B_{3,\text{cross}}$	the requested third output is an assigned input	$2q^2$

The inverse-private-output check gives $B_{\leftarrow \text{priv}}$, and the prepublication check gives $B_{\text{in-priv}}$. Numbered checks 1, 4, and 5 give B_h , $B_{\text{un}} \cup B_{\leftarrow 2}$, and B_{rev} , respectively. The insertion checks give the final four families. Thus no branch that sets the flag is omitted.

For $B_{\leftarrow 2}$, a fresh inverse input may satisfy Equation (3) with any earlier public record as E_1 . Prefix uniqueness limits the matches after the input is sampled, not the equalities tested before sampling, so we count at most q candidates per cycle. For $B_{3,\text{cross}}$, the k th cycle has at most $2k-1$ assigned records before insertion. Including the requested input itself therefore gives at most $2k \leq 2q$ candidates.

The one-cycle analysis identifies the pivot for each row. Families B_h , B_{un} , $B_{\leftarrow 2}$, and B_{rev} use the unsupplied coordinate of the newly public record, or its private output after promotion. Families $B_{\leftarrow \text{priv}}$ and $B_{\text{in-priv}}$ use the private output; when a fresh inverse input is sampled, the latter family uses that input instead. The two third-input families use the second-edge output, or the input of a unary inverse prefix. The two third-output families use the fresh value $F(x)$. In every case, the tested equality fixes that pivot to one value after conditioning on the other exposed information.

Summing the last column gives at most $(23q^2 + q)/2 \leq B_q := 12(q+1)^2$ candidate equalities.

The at most $2q$ table records exclude at most $4q$ coordinate values from a pivot; we round this to $4(q+1)$. Failed earlier tests exclude at most B_q further values. With

$$\Delta_q := N - 4(q+1) - B_q,$$

each candidate equality succeeds with conditional probability at most $1/\Delta_q$. A failed test excludes one candidate value and does not resample the pivot. A union bound gives

$$\Pr[\text{bad}] \leq \frac{B_q}{\Delta_q}. \quad (6)$$

Soundness and coupling. Consider a target query at x in G_M . After its first two permutation calls, E_1 and E_2 are public. Consider the cycle that published the later of the two, or their common record if $E_1 = E_2$. This cycle may precede the target query or occur during one of its first two calls. If it published E_1 after E_2 , check 5 set the bad flag. Otherwise it published E_2 with label $\rightarrow$, $\perp \rightarrow$, or

$\leftarrow$. Check 3 then attempted to insert Equation (4), or check 4 set the bad flag; an attempted insertion either succeeds or sets the flag. Unless **bad** occurs, the third edge is therefore in the table before the third call, which either repeats a public answer or promotes that internal record. The third call therefore returns $F(x) + x + c_4$, and the construction output is $F(x)$. Hence G_M and G_I give D the same target answers until **bad**. Their primitive-phase transcripts are identical by definition. Their post-switch permutation states may differ privately, but D cannot query those states.

We jointly couple G_R and G_M by using the same lazy permutation samples until a prefix is completed. On a good completion, the requested third input is unused. The corresponding input x to F is fresh. Query $F(x)$ and set $V := F(x) + x + c_4$. If V is an unused permutation output, use V as the real output too. Otherwise, set the bad flag and sample the real output uniformly from the unused outputs. This coupling gives the real output the required uniform marginal on unused outputs. If the coupled value meets an extra privacy or cross-coordinate restriction, one of the explicit bad events occurs. Otherwise the two games extend their tables by the same edge. Induction over permutation cycles therefore keeps their visible transcripts identical until **bad**.

The three games can be realized on one joint probability space whose visible transcripts agree unless **bad** occurs. For the fixed nondegenerate schedule, the coupling inequality and Equation (6) therefore give advantage at most B_q/Δ_q . Averaging over the random tape of D proves the same bound for a randomized D . Finally, independently sampled setup constants satisfy $c_2 = c_3$ with probability $1/N$. Averaging over the setup coins gives the claimed bound. $\square$

Proof of Corollary 7.2. The proof of Theorem 7.1 uses randomness of the public constants only to bound the event $c_2 = c_3$. Fixing a schedule with $c_2 \neq c_3$ removes that event. The simulator, the remaining bad-event bounds, and the final conservative estimate are unchanged. $\square$

Taken as a structural analysis, the one-, two-, and three-call results give a precise separation. The one- and two-call inverse obstructions survive the permutation-preprocessing restriction, whereas three calls achieve a birthday-order bound in that model. Whether three calls achieve standard indistinguishability under arbitrarily interleaved access remains open. The result also gives no claim about three rounds of the concrete AES round function.

Multiple instances and greater depth. Independent public schedules with a shared permutation do not remove the two-call obstruction: Corollary B.2 applies to any fixed instance. For three or more calls, cross-instance chain reuse requires a separate analysis. Multi-instance security and the single-instance family beyond three calls remain open. The repository records the formal multi-instance interface [1].

C Checkable Punctured Vectors: Construction and Proofs

This appendix develops the CPV application from [Section 8](#). We define the interface and tree protocol, prove correctness and semi-honest security, and compare the construction with earlier tree expansions.

C.1 Interface and Tree Construction

Fix a leaf length $\kappa \geq 1$ and vector length $n \geq 2$. Indices start at zero. Our interface is related to a punctured PRF (PPRF). A PPRF key k defines a κ -bit value $F(k, i)$ at every index $i \in [0, n)$. Puncturing k at i^* produces a key k^* that preserves every other value:

$$F(k, i) = F(k^*, i), \quad \forall i \in [0, n) \setminus \{i^*\}.$$

Given only k^* , the value $F(k, i^*)$ should be computationally indistinguishable from uniform.

Punctured PRFs support silent OT [8] and several post-quantum signatures based on symmetric assumptions [15, 16, 28, 14]. Related two-party abstractions are often called all-but-one OT.

Our target deliberately differs from the usual PPRF interface. Party P_0 supplies the punctured index i , and P_1 starts the functionality. Party P_1 receives a uniform vector r . Party P_0 receives every entry except r_i and can test guesses for that missing value. The interface omits pseudorandom hiding against equality tests because in the targeted OT and PCG applications the missing leaf masks an independent high-entropy value. [Lemma C.4](#) quantifies the remaining guessing probability.

Functionality $\mathcal{F}_{\text{cpv}}$:

The initialization is one-shot. The functionality waits for an input $i \in [0, n)$ from party P_0 and **start** from party P_1 . It then samples $r \leftarrow (\{0, 1\}^\kappa)^n$ and outputs $(r_0, \dots, r_{i-1}, 0^\kappa, r_{i+1}, \dots, r_{n-1})$ to P_0 and r to P_1 . It ignores checks before initialization and ignores subsequent initialization inputs. After initialization, on each input (check, v) with $v \in \{0, 1\}^\kappa$ from P_0 , it outputs 1 to P_0 if $v = r_i$ and otherwise outputs 0. It reveals no check output to P_1 .

Figure 5: Functionality $\mathcal{F}_{\text{cpv}}$ for checkable punctured vectors.

For the tree protocol, take $n = 2^D$ with $D \geq 1$. Let $G : \{0, 1\}^\kappa \rightarrow (\{0, 1\}^\kappa)^2$ denote the expansion rule that maps a parent seed to its two children. We use the GGM binary-tree organization [57]. The root is $s_{0,0}$. Expanding a node $s_{d-1,j}$ produces the children $s_{d,2j}$ and $s_{d,2j+1}$. For a puncture i , define the critical-path index at level d as

$$p_d := \lfloor i/2^{D-d} \rfloor.$$

The sibling $p_d \oplus 1$, where $\oplus 1$ flips the last bit of the index, is the co-path node at that level. The D co-path seeds determine every leaf except i .

The sum trick of [8] transfers each co-path seed without revealing the critical-path choice to P_1 . At level d , P_1 forms one aggregate of all left-child seeds and one aggregate of all right-child seeds. Through one 1-out-of-2 OT, P_0 receives the aggregate whose parity is opposite the d -th critical-path bit. After expanding the co-path seeds already known from the preceding level, P_0 subtracts their aggregate and recovers the one new co-path seed. Thus the protocol uses one OT per level.

Random-oracle construction. We consider the following protocol in the ideal-OT hybrid. Let $H : \{0, 1\}^\kappa \rightarrow \{0, 1\}^\kappa$ be a random oracle. We identify its domain with $\mathbb{Z}_{2^\kappa}$. Let $\boxplus$ and $\boxminus$ denote addition and subtraction in this group. Define

$$G(x) := (H(x), H(x) \boxplus x).$$

Figure 6 uses this expansion at every node. An arbitrary requested length can be padded to $n' := 2^{\lceil \log_2 n \rceil}$, in which case the cost is determined by n' , not n . We use a one-shot batch ideal-OT functionality. Party P_0 supplies $a_d \in \{0, 1\}$, and party P_1 supplies $m_{d,0}, m_{d,1} \in \mathbb{Z}_{2^\kappa}$, for each $d \in [1, D]$. The functionality waits for both inputs and then evaluates

$$\mathcal{F}_{\text{OT}}^D(P_0 : a_1, \dots, a_D; P_1 : (m_{d,0}, m_{d,1})_{d=1}^D) := (m_{d,a_d})_{d=1}^D$$

with output only to P_0 . It reveals no output to P_1 and ignores subsequent inputs.

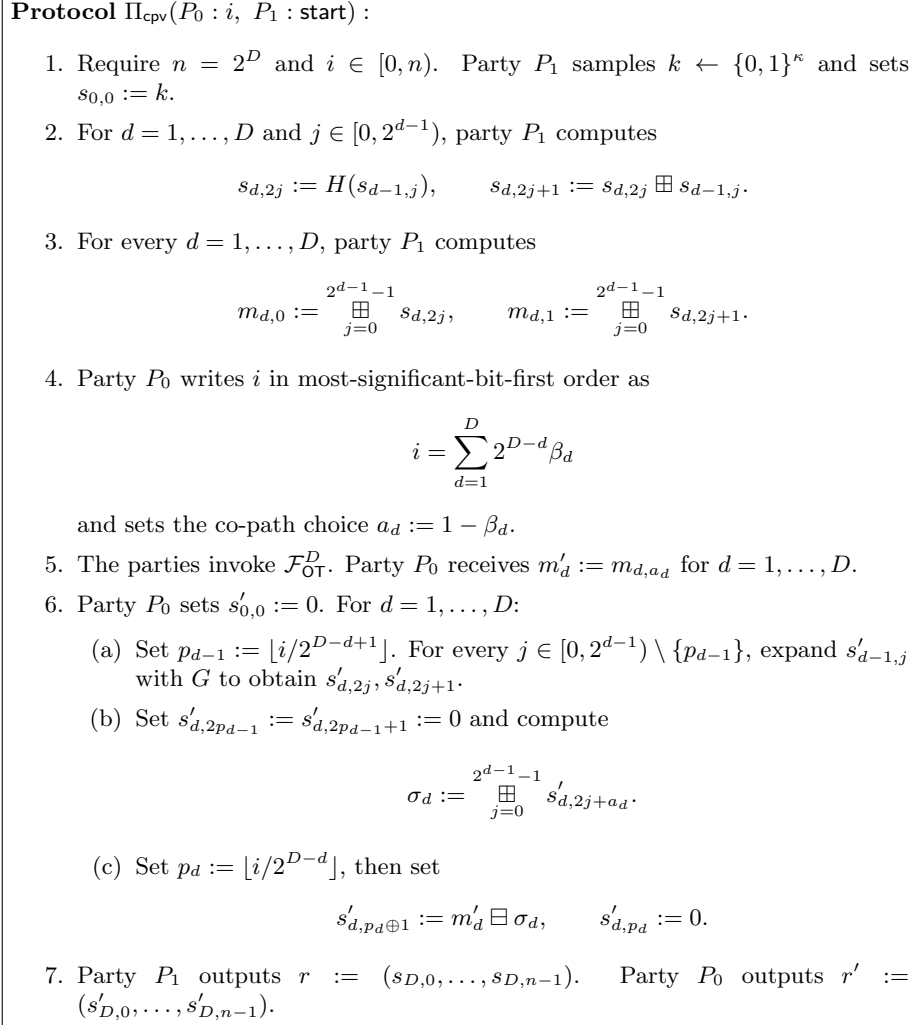


Figure 6: Protocol for checkable punctured vectors in the random-oracle model and ideal-OT hybrid.

C.2 Functional Correctness and Cost

Lemma C.1 (Functional correctness). *If both parties follow Π_{cpv} , then $r'_i = 0^\kappa$ and $r'_j = r_j$ for every $j \neq i$.*

Proof of Lemma C.1. We prove by induction on d that, after level d , party P_0 holds the correct seed at every position except p_d , where it holds zero. The claim holds at level zero. At level d , expanding every noncritical parent gives all children except the two children of p_{d-1} . Among those two, the child of parity $a_d = 1 - \beta_d$ is the co-path child. The OT value m'_d is the aggregate of all

children with that parity. The value σ_d contains every term in this aggregate except the co-path child, so $m'_d \boxplus \sigma_d$ recovers it. The other child is p_d and remains zero. The invariant at $d = D$ gives the claim. $\square$

Call count. Party P_1 expands all $n-1$ internal nodes. Party P_0 expands every internal node except the D nodes on the critical path, so it makes $n - D - 1$ calls. Each expansion uses one call to H , and the protocol uses D ideal OTs. For $\kappa = 128$, each oracle input and output has the type supplied by H_{128} . Replacing the oracle with H_{128} invokes the construction-level claim in [Table 1](#).

Checking the punctured point. Party P_0 can test a guess for the missing leaf using its known sibling. For a leaf pair with parent $s_{D-1,j}$, the protocol gives

$$r_{2j} = H(s_{D-1,j}), \quad r_{2j+1} = H(s_{D-1,j}) \boxplus s_{D-1,j}.$$

If $i = 2j$, party P_0 tests a guess v by checking

$$v = H(r_{2j+1} \boxplus v).$$

If $i = 2j + 1$, it instead checks

$$H(v \boxplus r_{2j}) = r_{2j}.$$

Either equation holds for $v = r_i$. For a wrong guess whose queried oracle input is still unassigned, the equation holds with probability $2^{-\kappa}$ over the fresh oracle answer. Thus the real test can have false positives, whereas the ideal check is exact. The security proof below reproduces the real test using ideal checks and simulated oracle answers.

C.3 Semi-honest Realization

Semi-honest security model. We consider one execution with one static semi-honest corruption. Fix κ , $n = 2^D$, the index $i \in [0, n)$, and an auxiliary input $z \in \{0, 1\}^*$. The corrupted party follows the protocol and may adaptively make auxiliary random-oracle queries before, during, or after its prescribed computation. Let q_H bound the number of these queries. The prescribed tree-expansion queries do not count toward q_H .

For $b \in \{0, 1\}$, let $\text{REAL}_b^{\mathcal{A}}(\kappa, n, i, z)$ denote the joint distribution of the corrupted party's view and the honest party's output when $\mathcal{A}$ controls P_b . The real execution gives both parties access to H and $\mathcal{F}_{\text{OT}}^D$. The view contains the corrupted party's input, z , its random coins, its oracle transcript, its OT transcript, and its prescribed output.

Let $\text{IDEAL}_b^{\mathcal{S}}(\kappa, n, i, z)$ denote the corresponding joint distribution when the parties use $\mathcal{F}_{\text{cpv}}$. The simulator receives the corrupted party's input and prescribed output before generating the simulated view. It simulates the random oracle with a table T . A repeated query returns the stored value. On a fresh query, the simulator either programs an undefined entry or stores and returns

an independent uniform value. If P_0 is corrupted, the simulator may also call the `check` interface. Both distributions are taken over the coins of the parties and adversary, the ideal functionality, and the random oracle or its simulation.

Write $\Delta(P, Q)$ for the statistical distance between distributions P and Q . Define

$$\epsilon_{\text{tree}} := \min \left\{ 1, \frac{\binom{n-1}{2} + (n-1)q_H}{2^\kappa} \right\}.$$

Theorem C.2 (Semi-honest realization). *Fix $b \in \{0, 1\}$. For every static semi-honest PPT adversary $\mathcal{A}$ that corrupts P_b and makes at most q_H auxiliary queries to H , there exists one PPT simulator $\mathcal{S}^{\mathcal{A}}$ such that, for every $\kappa \geq 1$, every power of two $n = 2^D \geq 2$ polynomially bounded in κ , every $i \in [0, n)$, and every auxiliary input z , we have*

$$\Delta(\text{REAL}_b^{\mathcal{A}}(\kappa, n, i, z), \text{IDEAL}_b^{\mathcal{S}^{\mathcal{A}}}(\kappa, n, i, z)) \leq \epsilon_{\text{tree}}.$$

The protocol runs in the $(\mathcal{F}_{\text{OT}}^D, H)$ -hybrid model. The simulator for a corrupted P_0 makes at most Dq_H calls to the `check` interface of $\mathcal{F}_{\text{cpv}}$.

We first record the distributional fact used in the simulation.

Lemma C.3 (Pairwise uniformity of tree labels). *Fix $\kappa \geq 1$ and $D \geq 1$. Let $\mathcal{Z} := \mathbb{Z}_{2^\kappa}$ and $n := 2^D$. Sample $s_{0,0} \leftarrow \mathcal{Z}$. For every internal position (d, j) with $0 \leq d < D$ and $0 \leq j < 2^d$, independently sample $h_{d,j} \leftarrow \mathcal{Z}$ and define*

$$s_{d+1,2j} := h_{d,j}, \quad s_{d+1,2j+1} := h_{d,j} \boxplus s_{d,j}.$$

Then the leaf vector $(s_{D,0}, \dots, s_{D,n-1})$ is uniform in $\mathcal{Z}^n$. Moreover, for any distinct tree positions $(d, j) \neq (e, k)$, the pair $(s_{d,j}, s_{e,k})$ is uniform in $\mathcal{Z}^2$.

Proof. For each expansion, the map

$$(s, h) \mapsto (h, h \boxplus s)$$

is a bijection, with inverse $(\ell, u) \mapsto (u \boxminus \ell, \ell)$. Applying these bijections level by level gives a bijection from the root and the $n - 1$ expansion samples to the n leaves. The leaf vector is therefore uniform in $\mathcal{Z}^n$.

Every node label can be reconstructed from the leaves below it by repeated use of

$$s_{d,j} = s_{d+1,2j+1} \boxminus s_{d+1,2j}.$$

It is consequently a signed sum of the leaves in its subtree, with every coefficient in $\{1, -1\}$. Consider two distinct positions. If their subtrees are disjoint, each label is uniform and the labels depend on disjoint sets of leaves. They are therefore independent. If one subtree strictly contains the other, write the ancestor label as

$$s_{d,j} = \sigma s_{e,k} \boxplus Z, \quad \sigma \in \{1, -1\},$$

where Z is a signed sum of leaves outside the descendant subtree. The value Z is uniform and independent of $s_{e,k}$. The map $(s_{e,k}, Z) \mapsto (s_{e,k}, \sigma s_{e,k} \boxplus Z)$ is bijective, so the two labels are jointly uniform. $\square$

Proof of Theorem C.2. Sampling game and query order.

The security game fixes the inputs and compares the completed view jointly with the honest output. The simulator receives its ideal output before generating that view. It first prepares the tree information described below, then runs $\mathcal{A}$ with its simulated party state. It exposes messages and answers auxiliary queries in the original protocol order, including queries before the prescribed tree computation.

We may evaluate the entire real tree before running $\mathcal{A}$. To see this, regard H as a uniformly sampled full function and sample the root independently. The tree is a deterministic function of these objects. Evaluating it early changes neither H nor any answer given to $\mathcal{A}$. The root component of P_1 's random tape is this sampled root; all other coins retain their original distribution. In particular, this argument allows an auxiliary query that depends on the root before tree expansion.

Compare this early evaluation with the sampling tree from Lemma C.3. Set **Bad** if two of its $n - 1$ internal-node labels coincide. Until the first such collision, each expansion in a fixed traversal visits a fresh oracle input. Couple its oracle answer with the independent expansion sample h . At a repeated input, the real oracle reuses its answer and the sampling tree takes a fresh sample; after this point the coupling may fail. If no collision occurs, the trees agree and $H(s) = h$ at every internal node. All remaining oracle entries are independent uniform values in both games. We can therefore couple the subsequent adaptive computation, including all auxiliary queries, with identical answers outside **Bad**.

Each simulator starts with an empty oracle table T and an unset bad flag.

We make each simulator total with a fallback procedure. When a bad condition below first occurs, the simulator sets **Bad** and invokes **Fallback**. This procedure retains existing entries of T , answers each fresh auxiliary query by lazy uniform sampling, fills every undefined simulated key or message with 0^κ , and returns a well-typed view. Its distribution after **Bad** is irrelevant to the distance bound.

In either corruption case, a repeated auxiliary query returns its entry in T . The simulator prepares its initial entries before running $\mathcal{A}$, as follows.

Corrupted P_1 . Consider first a corrupted P_1 . The simulator receives the ideal vector r . It reconstructs the tree from the leaves upward. For each sibling pair (ℓ, u) , it defines the parent as $u \boxminus \ell$. It sets the simulated root key k to the reconstructed root. Before programming a reconstructed parent s with left child ℓ , the simulator checks whether s has already appeared as a reconstructed parent or $T[s]$ is already defined. If either condition holds, it invokes **Fallback**. Otherwise, it records s and sets $T[s] := \ell$. It then computes P_1 's OT inputs from the reconstructed tree and runs $\mathcal{A}$ with k as the root component of the party's random tape. It samples the remaining coins as in the real execution. Fresh auxiliary queries outside T receive independent uniform answers, which it stores. The ideal OT reveals no receiver choice to P_1 , so the simulator needs no information about i .

Corrupted P_0 . Now consider a corrupted P_0 . The simulator receives i and

the $n - 1$ unpunctured leaves. Each co-path node is the root of a fully known subtree. The simulator reconstructs these subtrees upward with $s = u \boxplus \ell$. Before programming $T[s] := \ell$, it checks whether s is a repeated reconstructed parent or $T[s]$ is already defined. If either condition holds, it invokes **Fallback**. Otherwise, it records s and sets $T[s] := \ell$. At level d , the simulator sets the selected OT value to the $\boxplus$ -sum of the co-path seed and all other reconstructed seeds of parity a_d at that level. These values make P_0 recover exactly the co-path seeds and the ideal unpunctured leaves.

For a corrupted P_0 , it remains to simulate queries that might address the critical path. Write $\gamma_t := s_{t, p_t \oplus 1}$ for the known co-path node at level t . For each auxiliary query $H(x)$ not already in T , and each $d \in \{0, \dots, D-1\}$, the simulator treats x as a candidate $\hat{s}_d$ for the critical-path node at level d . It propagates this candidate toward the punctured leaf by defining, for $t = d + 1, \dots, D$,

$$\hat{s}_t := \begin{cases} \gamma_t \boxplus \hat{s}_{t-1}, & \text{if } \beta_t = 0, \\ \gamma_t \boxplus \hat{s}_{t-1}, & \text{if } \beta_t = 1. \end{cases}$$

The simulator submits $\hat{s}_D$ to the check interface for each candidate level. If exactly one check accepts, at level d , it programs

$$T[x] := \begin{cases} \gamma_{d+1} \boxplus x, & \text{if } \beta_{d+1} = 0, \\ \gamma_{d+1}, & \text{if } \beta_{d+1} = 1. \end{cases}$$

If no check accepts, it samples $T[x] \leftarrow \{0, 1\}^\kappa$. The simulator invokes **Fallback** if more than one check accepts; otherwise it returns $T[x]$. Each propagation step is bijective. The check for level d therefore accepts exactly when $x = s_{d, p_d}$. More than one check accepts only when two critical-path labels coincide.

Outside **Bad**, the known internal labels are distinct from all critical-path labels. Every fresh query at a critical-path label is recognized and receives its prescribed left child. Every other fresh query outside the initial table addresses a noninternal label and receives an independent uniform value. Thus the simulator implements the same oracle as conceptual programming of the entire sampling tree, even though it does not know the punctured leaf. The prescribed computation uses the already reconstructed subtrees, so only auxiliary queries require checks. There are at most Dq_H such checks.

Bad-event bound. The ideal vector is uniform. Reconstructing its conceptual full tree gives exactly the sampling distribution in [Lemma C.3](#), including the punctured leaf hidden from a corrupted P_0 . Each simulator bad condition implies that two internal labels of this tree coincide. Before $\mathcal{A}$ runs, T contains only reconstructed parents, so a predefined entry $T[s]$ is a repeated reconstructed parent. Two accepting checks identify equal critical-path labels at distinct levels. The simulator's fallback therefore occurs only in the event **Bad** of the sampling game. Pairwise uniformity and a union bound give

$$\Pr[\text{Bad}] \leq \min \left\{ 1, \frac{\binom{n-1}{2}}{2^\kappa} \right\} \leq \epsilon_{\text{tree}}.$$

Both simulators reproduce the sampling game outside **Bad**. The coupling above also makes the real view and honest output agree with those of that game outside the same event. The coupling inequality proves the theorem. The term $(n-1)q_H/2^\kappa$ in ϵ_{tree} is slack: the simulation in this completed-view model incurs no auxiliary-query loss. $\square$

C.4 Guessing through the Check Interface

Lemma C.4 (Guessing through the check interface). *Let $U \leftarrow \{0,1\}^\kappa$ be independent of an adversary's initial view. Suppose the adversary makes at most $q < 2^\kappa$ adaptive queries to $\text{check}_U(v) := [v = U]$ and then outputs U' . Then*

$$\Pr[U' = U] \leq \frac{q+1}{2^\kappa}.$$

Proof of Lemma C.4. Condition on the adversary's initial view and independent internal randomness. The value U remains uniform under this conditioning. Consider the transcript in which every check returns zero. Let $v_1, \dots, v_q$ be the queries on this transcript, padding the list if the adversary stops early, and let u' be its final output. If $U \notin \{v_1, \dots, v_q\}$, then the actual transcript is the all-zero transcript. The adversary then succeeds only when $U = u'$. Hence success requires $U \in \{v_1, \dots, v_q, u'\}$, a set of size at most $q+1$. Its conditional probability is at most $(q+1)/2^\kappa$. Averaging over the conditioned values gives the claimed bound. $\square$

In the targeted OT and PCG uses, the punctured leaf masks an independent high-entropy key or correlation, such as the OT/VOLE correlation Δ . The lemma shows that the check interface gives no shortcut beyond guessing that value. Therefore $\mathcal{F}_{\text{cpv}}$ suffices for these masking steps, and they do not require the stronger PPRF interface. An application that exposes another function of the missing leaf requires a separate analysis of that leakage.

C.5 Comparison with Earlier Tree Expansions

The conventional GGM construction implements G as a pseudorandom generator [57, 8]. Using two AES calls to produce the two κ -bit children costs approximately $2n$ AES calls for a full tree.

Boyle et al. [17] give a permutation-based construction with a weaker unpredictability target. Let P be a public permutation on $\{0,1\}^\kappa$ and define $H_{\text{DM}}(s) := P(s) + s$. Their two children are

$$G_{\text{EA}}(s) := (H_{\text{DM}}(s) + s, H_{\text{DM}}(s) \boxplus s) = (P(s), (P(s) + s) \boxplus s),$$

where $+$ is bitwise XOR and $\boxplus$ is addition in $\mathbb{Z}_{2^\kappa}$. They provide evidence for unpredictability in the random-invertible-permutation model but do not claim the stronger unpredictability notion achieved by their random-oracle construction. Hashing one side at the leaves gives a PPRF-style expansion at about $1.5n$ primitive calls; a full security analysis of the permutation-based construction remains open.

Half-Tree optimizations. Guo et al. [62] present correlated GGM trees for correlated OT (COT) and subfield VOLE (sVOLE). Their correlation-specific COT tree uses approximately n random-permutation calls. Their pseudorandom correlated GGM construction hashes the leaves to recover a PPRF-style output and costs approximately $1.5n$ calls. Its stronger interface hides the punctured leaf even from equality tests. Our construction instead states the equality test explicitly and realizes $\mathcal{F}_{\text{cpv}}$ with $n - 1$ calls for P_1 and $n - D - 1$ calls for P_0 . Thus it saves approximately one third of the primitive calls relative to the pseudorandom-leaf Half-Tree when $\mathcal{F}_{\text{cpv}}$ suffices for the application. We refer to [62] for the Half-Tree protocols, cost accounting, and proofs in the random-permutation model.

Cui et al. give the closest AES-hashing baseline for this application [28]. They construct an AES-based hash that satisfies circular correlation robustness and a new extractability property. They prove both properties in the ideal-cipher model. Their all-but-one vector commitment integrates the Half-Tree optimization. When the security parameter equals the block length, their hash reduces to the earlier MMO-style circular-correlation-robust (CCR) construction and uses one block-cipher invocation. Their result establishes that a specialized correlation-robust interface can support fast AES-based tree expansion.

Theorem C.2 shows that Π_{cpv} realizes $\mathcal{F}_{\text{cpv}}$ in the random-oracle and ideal-OT model. Its explicit simulators account for equality testing of the missing value. This smaller interface supports the stated high-entropy masking steps and avoids the leaf hashes required for PPRF-style outputs. A direct wall-clock comparison with the earlier protocols would require implementing the same complete protocol and is outside this paper’s scope.

D Family Constants and Test Vectors

This appendix lists the family constants and test vectors as byte strings. Bytes inside each 128-bit block use AES state order.

For the default H_{128} instance, the input

$$x = 000102030405060708090a0b0c0d0e0f$$

produces

$$H_{128}(x) = 87bc4c3694b34fc959eab45d84b0ea77.$$

Iteration $j \in \{0, \dots, 9\}$ of the wide construction uses RC_{2j} in lane zero and RC_{2j+1} in lane one.

k	RC_k
0	9d7b8175f0fec5b20ac020e64c708406
1	17f7082fa46b0f646ba0f388e1b4668b
2	1491029f609d02cf9884f2532dde0234
3	794f5bfda6cf3bb084f7b2ee6ead60e
4	447039be1ccdee798b447248cbb0cfcb
5	7b058a2bed35538db732906eeecdea7e
6	1bef4fda612741e2d07c2e5e438fc267
7	3b0bc71fe2fd5f6707cccaafb0d92429
8	ee65d4b9ca8fdbce97f86e6f1634dab
9	337e03ad4f402a5b64cdb7d484bf301c
10	0098f68d2e8b0269bf231794b90bccb2
11	8a2d9d5cc89eaa4a72556fdea67804fa
12	d49f12292e4ffa0e122a776b2b9fb4df
13	ee126abbae11d63236a249f44403a11e
14	a6eca89cc900965f8400054b884904af
15	ec93e527e3c7a2784f9c199dd85e0221
16	7301d482cd2e28b9b7c959a7f8aa3abf
17	6b7d3010d9eff23717b086610d706062
18	c69afcf65391c28143043021c245ca5a
19	3a94d136e892af2cbb686b223c972392

For H128tw, the published AES-counter stream key is

$$K_{tw} = \text{a5df71aff064d44dbbc6bde5dd1fc1b8}.$$

Epoch zero has

$$\begin{aligned} b_1^{(0)} &= \text{8a85c7ffab71111eaa9112b3f7024890}, \\ b_{11}^{(0)} &= \text{8a1f4e119de4881cce29658fa2c3b52b}. \end{aligned}$$

For the message

$$\text{000102030405060708090a0b0c0d0e0f},$$

the epoch-zero outputs at tweaks 31, 32, and 255 are respectively

$$\begin{aligned} &\text{73ef410f28eeb75ea67be04106222f0d}, \\ &\text{385350b8fc914b9ebb97e0650674f161}, \\ &\text{afd297e0d4ca16982d019f6a48d30f59}. \end{aligned}$$

Tweaks 31 and 32 lie on opposite sides of a cache-window boundary (see below). For epoch one, the same message at tweak zero produces

$$\text{71333c97beeb241ba35b1d55cf22441c}.$$

Machine-readable vectors also cover the zero message, the maximum tweak, and the maximum 64-bit epoch.

An optimized implementation may cache eleven effective constants for each tweak in an aligned window $w, w+1, \dots, w+31$, where $w \in \{0, 32, \dots, 224\}$. The table contains 352 blocks, or 5,632 bytes. Its cache identity is (a, w) . Constant generation and field arithmetic occur only during epoch or window setup.

For the wide input

000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f,

the 256-bit output is

4275eeb2228d9cf88b935fa5352539aa88e51ccf3db7f9ed85800af877a88c5d.

Truncating the output to 128 bits returns the first displayed block. Machine-readable vectors include zero, all-ones, and split-pattern inputs.

E Adversarially Chosen Constants

The constructions in this paper fix their constants through deterministic derivation rules. This appendix removes that restriction as a diagnostic: a parameter generator may choose individual round constants after selecting inputs. The resulting attacks motivate the fixed derivation rules. They do not apply to the specified schedules.

We first isolate the one-block mechanism. Let R be the keyless AESENC map and write the recurrence as

$$q_1 = x + c_1, \quad q_{j+1} = R(q_j) + x + c_{j+1} \quad (j = 1, \dots, 10).$$

Proposition E.1 (Programming the penultimate constant). *Fix $c_1, \dots, c_9, c_{11}$ and distinct inputs x, x' . Let q_9, q'_9 be their states after the fixed prefix, set $D = x + x'$, and define*

$$\delta := R(q_9) + R(q'_9) + D.$$

If some $b \in \{0, 1\}^{128}$ satisfies

$$R(b) + R(b + \delta) = D,$$

then some choice of c_{10} makes the complete outputs on x and x' equal.

Proof. Set $c_{10} := b + R(q_9) + x$. The two states entering the final layer are then $q_{10} = b$ and $q'_{10} = b + \delta$. Their final output difference is

$$R(b) + R(b + \delta) + D = 0.$$

The fixed value c_{11} cancels from this difference. □

The premise is an exact AES differential-fiber condition. For the published prefix, the accompanying search finds a compatible H_{128} pair after 61,429 trials. Changing only c_{10} then completes the collision. For a fixed H_{128}^{tw} tweak, the public tweaks translate the constants, so the same method targets one slice by changing its base block b_{10} . For the wide construction, changing the two 256-bit blocks injected in layers 8 and 9 gives an analogous complete collision with about 2^{16} trials per lane. Each result exhibits one collision for one schedule;

none gives a procedure for finding further collisions after the schedule has been fixed.

A separate choice of constants creates a persistent structural weakness in the wide construction. Write its 256-bit initialization and layer constants as

$$K_0 := \text{iv}_0 \parallel \text{iv}_1, \quad K_{j+1} := \text{RC}_{2j} \parallel \text{RC}_{2j+1} \quad (j = 0, \dots, 9).$$

Thus K_{j+1} is the block added in iteration j , before any column permutation of that layer. Define the 32-dimensional subspace

$$U := \{w \parallel w \parallel w \parallel w \parallel w \parallel w \parallel w \parallel w : w \in \{0, 1\}^{32}\}.$$

The parallel AES layer and the wide column permutation preserve U . If the composite initialization block K_0 and layer blocks $K_1, \dots, K_9$ belong to one affine coset $a + U$, then every input in $a + U$ starts in U and remains there through layer 9. Layer 10 therefore gives an output in $\text{mix}_{256}(a + K_{10}) + U$. This gives a one-query membership distinguisher and restricts the relevant map to 32 bits. The accompanying verification code also exhibits a complete collision in this restricted map. The mechanism is the wide analogue of the symmetry exploited against the first Haraka proposal [67].

Under independent uniform sampling, condition on K_0 . Each remaining prefix block lies in $K_0 + U$ with probability 2^{-224} , independently of the other blocks. The complete condition therefore has probability

$$2^{-224 \cdot 9} = 2^{-2016};$$

K_{10} is unrestricted. The published wide schedule violates $K_j + K_0 \in U$ for every $j = 1, \dots, 9$. Moreover, the planted-collision schedules choose their constants as a function of a collision witness. They do not identify a significant-density class of weak random schedules.

The published one-block schedule also excludes Jean’s equal-column symmetry [67]. Let V_{sym} be the 32-dimensional subspace of AES states whose four columns are equal. The keyless AES round preserves V_{sym} . A chain beginning in this subspace remains there under every key addition only if

$$c_1 + c_j \in V_{\text{sym}} \quad (j = 2, \dots, 11).$$

The published instance-0 schedule violates every required inclusion. This test excludes the specific symmetry; it does not classify all invariant-subspace attacks.

References

- [1] Peter Rindal. An input-fed AES-round family for fixed-length hashing: accompanying code, validation artifacts, and supplementary analyses, 2026. <https://github.com/ladnir/aesHash>. Supplementary guide: [docs/supplementary/README.md](https://github.com/ladnir/aesHash/blob/main/docs/supplementary/README.md).

- [2] Elena Andreeva, Bart Mennink, and Bart Preneel. On the indistinguishability of the Grøstl hash function. In *SCN*, 2010.
- [3] National Institute of Standards and Technology. Advanced Encryption Standard (AES), FIPS 197, 2001.
- [4] Nicky Mouha. Review of the Advanced Encryption Standard. NISTIR 8319, National Institute of Standards and Technology, 2021.
- [5] National Institute of Standards and Technology. SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions, FIPS 202, 2015.
- [6] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. On the indistinguishability of the sponge construction. In *EUROCRYPT*, 2008.
- [7] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. The making of Keccak. *Cryptologia*, 38(1):26–60, 2014.
- [8] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. Efficient two-round OT extension and silent non-interactive secure computation. In *CCS*, 2019.
- [9] Bart Preneel, René Govaerts, and Joos Vandewalle. Hash functions based on block ciphers: A synthetic approach. In *CRYPTO*, pages 368–378, 1993.
- [10] John Black, Phillip Rogaway, and Thomas Shrimpton. Black-box analysis of the block-cipher-based hash-function constructions from PGV. In *CRYPTO*, pages 320–335, 2002.
- [11] Hidenori Kuwakado and Masakatu Morii. Indistinguishability of single-block-length and rate-1 compression functions. *IEICE Transactions on Fundamentals*, 90-A(10):2301–2308, 2007.
- [12] Brent Carmer and Mike Rosulek. Linicrypt: A model for practical cryptography. In *CRYPTO*, Part III, pages 416–445, 2016.
- [13] Zahra Javar and Bruce M. Kapron. Linicrypt in the ideal cipher model. IACR Cryptology ePrint Archive, Report 2024/706, 2024.
- [14] The FAEST Team. FAEST: Algorithm specifications. Technical report, NIST PQC additional signatures submission, 2023.
- [15] Carsten Baum, Lennart Braun, Cyprien Delpech de Saint Guilhem, Michael Klooß, Emmanuela Orsini, Lawrence Roy, and Peter Scholl. Publicly verifiable zero-knowledge and post-quantum signatures from VOLE-in-the-head. In *CRYPTO*, 2023.
- [16] Carsten Baum, Ward Beullens, Shibam Mukherjee, Emmanuela Orsini, Sebastian Ramacher, Christian Rechberger, Lawrence Roy, and Peter Scholl. One tree to rule them all: Optimizing GGM trees and OWFs for post-quantum signatures. IACR Cryptology ePrint Archive, 2024.

- [17] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Nicolas Resch, and Peter Scholl. Correlated pseudorandomness from expand-accumulate codes. In *CRYPTO*, 2022.
- [18] Mihir Bellare, Viet Tung Hoang, Sriram Keelveedhi, and Phillip Rogaway. Efficient garbling from a fixed-key blockcipher. In *IEEE Symposium on Security and Privacy*, 2013.
- [19] Srimanta Bhattacharya and Mridul Nandi. Full indifferentiable security of the XOR of two or more random permutations. In *EUROCRYPT*, 2018.
- [20] Ritam Bhaumik, Nilanjan Datta, Avijit Dutta, Ashwin Jha, Sougata Mandal, Bart Mennink, Hrithik Nandi, and Yaobin Shen. How to build a short-input random oracle from public random permutations. In *EUROCRYPT*, pages 394–423, 2026.
- [21] Tim Beyne, Gregor Leander, and Immo Schütt. Pairwise independence of AES-like block ciphers. In *EUROCRYPT*, pages 181–210, 2026.
- [22] Christina Boura, Patrick Derbez, and Margot Funk. Alternative key schedules for the AES. In *ACNS*, pages 485–506, 2024.
- [23] Guido Bertoni, Joan Daemen, Michael Peeters, and Gilles Van Assche. Sponge functions. ECRYPT Hash Workshop, 2007.
- [24] Alex Biryukov and David Wagner. Slide attacks. In *FSE*, 1999.
- [25] Alex Biryukov and David A. Wagner. Advanced slide attacks. In *EUROCRYPT*, 2000.
- [26] Jean-Sebastien Coron, Yevgeniy Dodis, Cecile Malinaud, and Prashant Puniya. Merkle-Damgård revisited: How to construct a hash function. In *CRYPTO*, 2005.
- [27] Shan Chen, Rodolphe Lampe, Jooyoung Lee, Yannick Seurin, and John Steinberger. Minimizing the two-round Even–Mansour cipher. In *CRYPTO*, 2014.
- [28] Hongrui Cui, Chun Guo, Xiao Wang, Chenkai Weng, Kang Yang, and Yu Yu. AES-based CCR hash with high security and its application to zero-knowledge proofs. IACR Cryptology ePrint Archive, Paper 2024/1271, 2024.
- [29] Benoîtogliati and Yannick Seurin. On the provable security of the iterated Even–Mansour cipher against related-key and chosen-key attacks. In *EUROCRYPT*, pages 584–613, 2015.
- [30] Seung Geol Choi, Jonathan Katz, Ranjit Kumaresan, and Hong-Sheng Zhou. On the security of the “Free-XOR” technique. In *TCC*, pages 39–53, 2012.

- [31] Takanori Isobe, Ryoma Ito, Fukang Liu, Kazuhiko Minematsu, Motoki Nakahashi, Kosei Sakamoto, and Rentaro Shiba. Areion: Highly-efficient permutations and its applications to hash functions for short input. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2023(2):115–154, 2023.
- [32] Emanuele Bellini, Alessandro De Piccoli, Mattia Formenti, David Gerault, Paul Huynh, Simone Pelizzola, Sergio Polese, and Andrea Visconti. Differential cryptanalysis with SAT, SMT, MILP, and CP: A detailed comparison for bit-oriented primitives. IACR Cryptology ePrint Archive, 2024.
- [33] Yu Long Chen and Stefano Tessaro. Better security-efficiency trade-offs in permutation-based two-party computation. In *ASIACRYPT*, 2021.
- [34] Orr Dunkelman, Shibam Ghosh, Nathan Keller, Gaetan Leurent, Avichai Marmor, and Victor Mollimard. Partial sums meet FFT: Improved attack on 6-round AES. In *EUROCRYPT*, 2024.
- [35] Mathieu Degré, Patrick Derbez, and André Schrottenloher. Simplified meet-in-the-middle preimage attacks on AES-based hashing. *IACR Communications in Cryptology*, 2(4):9, 2026.
- [36] Xiaoyang Dong, Jian Guo, Shun Li, and Phuong Pham. Triangulating rebound attack on AES-like hashing. In *CRYPTO*, pages 94–124, 2022.
- [37] Christoph Dobraunig, Maria Eichlseder, and Florian Mendel. Cryptanalysis of Simpira v1. In *SAC*, pages 284–298, 2016.
- [38] Yevgeniy Dodis, Thomas Shrimpton, Martijn Stam, and Stefano Tessaro. Indifferentiability of confusion-diffusion networks. In *CRYPTO*, 2016.
- [39] Yevgeniy Dodis, Thomas Ristenpart, John Steinberger, and Stefano Tessaro. To hash or not to hash again? (In)differentiability results for H^2 and HMAC. In *CRYPTO*, 2012.
- [40] Wei Dai, Viet Tung Hoang, and Stefano Tessaro. Information-theoretic indistinguishability via the chi-squared method. In *CRYPTO*, 2017.
- [41] Joan Daemen, Lars R. Knudsen, and Vincent Rijmen. The block cipher SQUARE. In *FSE*, 1997.
- [42] Itai Dinur. Combining outputs of a random permutation: New constructions and tight security bounds by Fourier analysis. IACR Cryptology ePrint Archive, 2024.
- [43] Yevgeniy Dodis, Thomas Shrimpton, Martijn Stam, and Stefano Tessaro. Provable security of substitution-permutation networks. In *CRYPTO*, 2017.
- [44] Daniel Masny and Peter Rindal. Endemic oblivious transfer. IACR Cryptology ePrint Archive, 2019.

- [45] Niels Ferguson, John Kelsey, Stefan Lucks, Bruce Schneier, Michael Stay, David A. Wagner, and Doug Whiting. Improved cryptanalysis of Rijndael. In *FSE*, 2000.
- [46] Pooya Farshim and Gordon Procter. The related-key security of iterated Even–Mansour ciphers. In *FSE*, pages 342–363, 2015.
- [47] Zhenzhen Bao, Xiaoyang Dong, Jian Guo, Zheng Li, Danping Shi, Siwei Sun, and Xiaoyun Wang. Automatic search of meet-in-the-middle preimage attacks on AES-like hashing. In *EUROCRYPT*, pages 771–804, 2021.
- [48] Ralph Ankele, Christoph Dobraunig, Jian Guo, Eran Lambooj, Gregor Leander, and Yosuke Todo. Zero-correlation attacks on tweakable block ciphers with linear tweakkey expansion. In *FSE*, 2019.
- [49] Achiya Bar-On, Orr Dunkelman, Nathan Keller, and Ariel Weizman. DLCT: A new tool for differential-linear cryptanalysis. In *EUROCRYPT*, 2019.
- [50] Zhenzhen Bao, Jian Guo, Danping Shi, and Yi Tu. Superposition meet-in-the-middle attacks: Updates on fundamental security of AES-like hashing. In *CRYPTO*, pages 64–93, 2022.
- [51] Shiyao Chen, Jian Guo, Eik List, Danping Shi, and Tianyu Zhang. Diving deep into the preimage security of AES-like hashing. In *EUROCRYPT*, pages 398–426, 2024.
- [52] Shiyao Chen, Jian Guo, Eik List, Danping Shi, and Tianyu Zhang. Scrutinizing the security of AES-based hashing and one-way functions. In *ASIACRYPT*, pages 157–188, 2025.
- [53] Praveen Gauravaram, Lars R. Knudsen, Krystian Matusiewicz, Florian Mendel, Christian Rechberger, Martin Schl  ffer, and S  ren S. Thomsen. Gr  stl: A SHA-3 candidate. Dagstuhl Seminar Proceedings, 2009.
- [54] Shay Gueron and Nicky Mouha. Simpira v2: A family of efficient permutations using the AES round function. In *ASIACRYPT*, 2016.
- [55] Chun Guo and Dongdai Lin. Indifferentiability of 3-round Even-Mansour with random oracle key derivation. IACR Cryptology ePrint Archive, 2016.
- [56] Chun Guo, Lei Wang, and Dongdai Lin. Impossibility of indifferentiable iterated blockciphers from 3 or less primitive calls. In *EUROCRYPT*, 2023.
- [57] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. In *FOCS*, 1984.
- [58] David Gerault, Pascal Lafourcade, Marine Minier, and Christine Solnon. Computing AES related-key differential characteristics with constraint programming. *Artificial Intelligence*, 2020.

- [59] Chun Guo, Jonathan Katz, Xiao Wang, and Yu Yu. Efficient and secure multiparty computation from fixed-key block ciphers. In *IEEE Symposium on Security and Privacy*, 2020.
- [60] Chun Guo, Xiao Wang, Kang Yang, and Yu Yu. On tweakable correlation robust hashing against key leakages. IACR Cryptology ePrint Archive, Paper 2024/163, 2024.
- [61] Aldo Gunesing, Ritam Bhaumik, Ashwin Jha, Bart Mennink, and Yaobin Shen. Revisiting the indistinguishability of the sum of permutations. In *CRYPTO*, 2023.
- [62] Xiaojie Guo, Kang Yang, Xiao Wang, Wenhao Zhang, Xiang Xie, Jiang Zhang, and Zheli Liu. Half-tree: Halving the cost of tree expansion in COT and DPF. In *EUROCRYPT*, 2023.
- [63] Shoichi Hirose, Jeffery Kilian, and Thomas Tessaro. Security of permutation-based hash constructions. In *ASIACRYPT*, 2011.
- [64] Tetsu Iwata. New blockcipher modes of operation with beyond the birthday bound security. In *FSE*, 2006.
- [65] Tetsu Iwata, Bart Mennink, and Damian Vizár. CENC is optimally secure. IACR Cryptology ePrint Archive, Report 2016/1087, 2016.
- [66] Jeremy Jean, Ivica Nikolic, and Thomas Peyrin. Tweaks and keys for block ciphers: The TWEAKEY framework. In *ASIACRYPT*, 2014.
- [67] Jérémy Jean. Cryptanalysis of Haraka. *IACR Transactions on Symmetric Cryptology*, 2016(1):1–12, 2016.
- [68] Jeremy Jean and Ivica Nikolic. Efficient design strategies based on the AES round function. In *FSE*, 2016.
- [69] Stefan Kolbl, Martin M. Lauridsen, Florian Mendel, and Christian Rechberger. Haraka v2: Efficient short-input hashing for post-quantum applications. *IACR Transactions on Symmetric Cryptology*, 2016(2):1–29, 2016.
- [70] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In *ICALP*, pages 486–498, 2008.
- [71] Lars R. Knudsen and Vincent Rijmen. Known-key distinguishers for some block ciphers. In *ASIACRYPT*, 2007.
- [72] Lars R. Knudsen and David Wagner. Integral cryptanalysis. In *FSE*, 2002.
- [73] Fukang Liu, Takanori Isobe, Willi Meier, and Kosei Sakamoto. Weak keys in reduced AEGIS and Tiaoxin. IACR Cryptology ePrint Archive, 2021.
- [74] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In *CRYPTO*, 2003.

- [75] Charlotte Lefevre and Bart Mennink. Permutation-based hashing beyond the sponge bound. IACR Cryptology ePrint Archive, 2024.
- [76] Charlotte Lefevre. Indifferentiability of the sponge construction with a restricted number of message blocks. *IACR Transactions on Symmetric Cryptology*, 2023(1):224–243, 2023.
- [77] Joelle Lim, Derrick Ng, and Ruth Ng. SoK: Security Evaluation of SBox-Based Block Ciphers. Cryptology ePrint Archive, Report 2022/1098, 2022.
- [78] Mitsuru Matsui. Linear Cryptanalysis Method for DES Cipher. In *EUROCRYPT*, pages 386–397, 1993.
- [79] Kaisa Nyberg. Linear Approximation of Block Ciphers. In *EUROCRYPT*, pages 439–444, 1995.
- [80] Ueli Maurer, Renato Renner, and Clemens Holenstein. Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In *TCC*, 2004.
- [81] Avradip Mandal, Jacques Patarin, and Yannick Seurin. On the public indifferentiability and correlation intractability of the 6-round Feistel construction. In *TCC*, pages 285–302, 2012.
- [82] Avradip Mandal, Jacques Patarin, and Valerie Nachev. Indifferentiability beyond the birthday bound for the XOR of two public random permutations. In *INDOCRYPT*, 2010.
- [83] Ofri Nevo, Ni Trieu, and Avishay Yanai. Simple, fast malicious multiparty private set intersection. In *CCS*, 2021.
- [84] Ivica Nikolic. Tiaoxin-346. Submission to the CAESAR competition, 2014.
- [85] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. PSI from Paxos: Fast, malicious private set intersection. In *EUROCRYPT*, 2020.
- [86] Joan Daemen and Vincent Rijmen. The block cipher Rijndael. In *CARDIS*, 1998.
- [87] Joan Daemen and Vincent Rijmen. AES proposal: Rijndael. AES Algorithm Submission, 1999.
- [88] Lawrence Roy. SoftSpokenOT: Quieter OT extension from small-field silent VOLE in the Minicrypt model. In *CRYPTO*, 2022.
- [89] Srinivasan Raghuraman and Peter Rindal. Blazing fast PSI from improved OKVS and subfield VOLE. In *CCS*, 2022.
- [90] Peter Rindal and Phillipp Schoppmann. VOLE-PSI: Fast OPRF and circuit-PSI from vector-OLE. In *EUROCRYPT*, 2021.

- [91] Stefan Lucks. The sum of PRPs is a secure PRF. In *EUROCRYPT*, 2000.
- [92] Takuro Shiraya, Subhadeep Banik, Tatsuya Ishikawa, Ryoma Ito, Mostafizar Rahman, Kosei Sakamoto, Atsushi Tanaka, Shion Utsumi, and Takanori Isobe. Exploring SHA instructions and its application to AES-based schemes. *IACR Communications in Cryptology*, 2(3):28, 2025.
- [93] André Schrottenloher and Marc Stevens. Simplified MITM modeling for permutations: New (quantum) attacks. In *CRYPTO*, Part III, pages 717–747, 2022. Full version: IACR Cryptology ePrint Archive, Report 2022/189.
- [94] Kodai Taiyama, Kosei Sakamoto, Ryoma Ito, Kazuma Taka, and Takanori Isobe. Key collisions on AES and its applications. IACR Cryptology ePrint Archive, 2024.
- [95] Ivan Tjuawinata, Tao Huang, and Hongjun Wu. Cryptanalysis of Simpira v2. In *ACISP*, pages 384–401, 2017.
- [96] Yosuke Todo. Structural evaluation by generalized integral property. In *EUROCRYPT*, 2015.
- [97] Yosuke Todo and Masakatu Morii. Bit-based division property and application to Simon family. In *FSE*, 2016.
- [98] Zejun Xiang, Wentao Zhang, Zhenzhen Bao, and Dongdai Lin. Applying MILP method to searching integral distinguishers based on division property for 6 lightweight block ciphers. In *ASIACRYPT*, 2016.
- [99] Zahra Eskandari, Andreas Brasen Kidmose, Stefan Kölbl, and Tyge Tiessen. Finding integral distinguishers with ease. In *SAC*, 2018.
- [100] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, 2001.
- [101] Hongjun Wu and Bart Preneel. AEGIS: A fast authenticated encryption algorithm. Submission to the CAESAR competition, 2014.
- [102] Christoph Dobraunig and Eik List. Impossible-differential and boomerang cryptanalysis of round-reduced KIASU-BC. In *ASIACRYPT*, 2017.
- [103] Shanjie Xu, Qi Da, and Chun Guo. Chosen-key secure Even–Mansour cipher from a single permutation. *IACR Transactions on Symmetric Cryptology*, 2023(1):244–287, 2023.
- [104] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole: Reducing data transfer in garbled circuits using half gates. In *EUROCRYPT*, 2015.
- [105] Phillip Rogaway and John Steinberger. Constructing cryptographic hash functions from fixed-key blockciphers. In *CRYPTO*, pages 433–450, 2008. <https://web.cs.ucdavis.edu/~rogaway/papers/lp.pdf>.