Cryptanalysis of 7-Round AES via the Algebraic Structure of its S-box

Milad Nasr and Nicholas Carlini

Anthropic

Abstract. The AES S-Box is not a random permutation; we show that its algebraic structure allows for improved attacks on 7 rounds in the single-key setting. We focus our efforts on extending a 2^{105} chosen plaintext and 2^{99}-time algorithm designed by Derbez, Fouque, and Jean [DFJ13]. Our core contribution is an algebraic *Möbius bridge* that exploits the invert-then-affine-transform structure of the AES S-box to eliminate one of the nine guessed key bytes required by [DFJ13]. This immediately leads to a factor-of-256 reduction in the number of key guesses required. But calculating the necessary Möbius Transform is not free, and the work required to perform the naïve calculation cancels the savings. We therefore present several improvements that reduce the time required to calculate this transform; depending on the exact methods used for accounting, we reduce the attack's runtime to between $2^{89.3}$ and $2^{91.4}$, at the same data complexity. This paper presents these improvements along with the computational techniques we used to help us empirically gain confidence in the correctness of our findings, something that was particularly important because these improvements were discovered autonomously via a large language model fully end to end. We conclude by remarking on the future of LLM-assisted research in cryptography and computer security more generally.

1 Introduction

The Advanced Encryption Standard (AES) [DBN+01], standardized by NIST in 2001, now underlies much of modern data and communication security. Since its design nearly thirty years ago [DR98], the full cipher has remained robust: the best single-key attacks (bicliques [BKR11]) beat brute force by only about two bits. Even reduced-round AES has remained relatively secure: at $D{=}2^{105}$, the best attack on seven rounds of AES remains a 2^{99} time attack due to [DFJ13] published in 2013.

In our paper we improve on this result. Our improvement continues this line of work begun by Demirci and Selçuk [DS08], who observed that the function performed by four AES rounds when restricted to a subset of its bytes is determined by only 25 parameter bytes. They show how a precomputed 2^{200}-entry table over those parameters can be used to mount a meet-in-the-middle attack on 7- and 8-round AES-256. Dunkelman, Keller, and Shamir [DKS10] introduced two key ideas to improve on this attack: a *multiset* fingerprint that is invariant to one guessed key byte on the input side, and a *differential enumeration* that shrinks the table from 2^{200} to 2^{127} entries — the first attack on 7-round AES-128 below exhaustive search, at 2^{116}. Derbez, Fouque, and Jean [DFJ13] tightened the parameter count to 10 bytes and rebalanced the differential, yielding an attack that requires 2^{105} chosen plaintexts, 2^{99} work, and 2^{90} storage.

Our contribution. Our primary contribution is the concept of a *Möbius Bridge*. Whereas [DKS10] showed how to construct a fingerprint that is invariant to the key byte *above* the meet-in-the-middle hashtable, we show how to construct a fingerprint that is also invariant to the key byte *below* the meet-in-the-middle hashtable. We do this by

exploiting the algebraic structure of the AES S-box, constructing a fingerprint algorithm that is invariant to affine transformations. This removes the need to guess this key byte, and so directly reduces the number of key guesses the attack must make by a factor of 256.

Our first fingerprint realizes this invariance algebraically, by forming ratios of power sums that cancel the unknown byte (§4.1). Unfortunately, naïve computation of this fingerprint is computationally expensive. We therefore introduce three implementation techniques — a packed power table (§A.1), a Gray-code walk over DDT solution choices (§A.2), and an XOR-separable S-box cache (§A.3) — that together bring the per-entry cost from $\approx 2^{19}$ to $\approx 2^{8.6}$ lookups. We additionally describe a second fingerprint, χ-canonicalization (§4.2), that achieves the same invariance by orbit canonicalization rather than algebraic elimination and is cheaper to evaluate by construction. Together, these improvements reduce the runtime of the attack (at a 2^{105} data complexity) down to between $2^{89.3}$ and $2^{91.4}$—depending on the exact methods used to account for the runtime of the algorithm.

These improvements were all discovered completely autonomously by a large language model. The remainder of this paper is thus dedicated to describing how we validated these findings through computational means. The Möbius transform is relatively straightforward to audit, and we are extremely confident it is correct; but because the final end product is an attack that runs in $\geq 2^{89}$ time we have no hope of verifying it by actually running it end-to-end. We therefore run a number of smaller scale computational experiments to help us gain confidence in the result (§5). This new paradigm—where the role of humans is primarily in verifying the output of a language model—feels uncomfortable; we conclude by commenting on what this shift might mean for cryptography and security research more broadly.

2 Overview and intuition

This is a relatively technical paper. In this section we provide a brief high level overview of our attack and the intuition for how it works, targeted at security researchers who may not necessarily be experts in cryptanalysis.

AES. The AES encryption algorithm is a 128-bit cipher[1] that is built by repeating ten identical rounds. Each round operates on a 128-bit input and transforms it by applying a non-linear lookup table (the "S-box") to each of the 16 8-bit bytes, and then undergoes two linear operations: one permutes the bytes of the state, and the other mixes each group of four bytes together; after this, the encryption key is mixed in via an xor operation. This process repeats for all ten rounds.

Meet-in-the-middle attacks on AES. We extend on a line of work developing "meet-in-the-middle" attacks [DS08]. In a traditional meet-in-the-middle attack, the attacker trades off time for space by running some algorithm that guesses key bytes and then uses this guessed key to encrypt (starting from the top of the cipher) and decrypt (starting from the bottom) some messages, and then checks for a collision that only occurs when the valid key has been guessed.

The AES meet-in-the-middle attack is slightly more complex. It turns out that applying 4 rounds of AES doesn't act as an ideal pseudo-random function over the 128-bit state. Specifically, take a structured set of 256 inputs that vary only by taking all possible values in the first byte, and encrypt each of these with four rounds of AES, to obtain a 256-tuple by taking the first byte of each of the resulting encryptions. The function that determines this 256-tuple could in principle be any one of $2^{8 \cdot 256}$ possible sequences. But [DS08] prove

[1] AES also comes in 192- and 256-bit variants. We focus on the 128-bit variant in this paper.

that actually only a small fraction of these sequences are possible over four rounds of AES, because the internal state is determined by only 25 parameter bytes. Call $\mathcal{T}$ the set of valid sequences; the attacker can enumerate and store this set offline. Followup work is able to significantly shrink the size of this table, a requirement to make the attack practical on 128 bit ciphers. We omit here a very important observation by [DKS10] who are only able to shrink the size of the lookup table by using techniques from differential cryptanalysis.

Suppose now that the attacker guesses a small number of key bytes at the top and bottom of the cipher—just enough to peel off the outermost rounds and expose the four-round core. With these guesses, the attacker can compute the 256-tuple from the actual ciphertexts and check whether it appears in the pre-computed $\mathcal{T}$. A match probabilistically guarantees the guessed key bytes are correct, and no match provably guarantees the key bytes are incorrect.

Eliminating one key-byte guess. In [DKS10], the authors show that one of the key bytes the attacker needs to guess can be eliminated. Specifically, it turns out that one of these bytes only operates as a permutation on the 256-tuple. And so if we can construct a fingerprint that is invariant to this reordering, we no longer need to guess that byte at all. The authors show that a *multiset* (where we record the frequency of each value instead of the ordered sequence) is exactly such a fingerprint, since it is unchanged by any permutation of the inputs. This contribution alone reduces the runtime of the attack by a factor of 2^8.

Our contribution. We show that it is possible to construct an even more elaborate fingerprint that eliminates the need to guess one more key byte, and then show how to compute this fingerprint sufficiently efficiently that we can obtain a factor of $2^{7.6}$ to $2^{9.7}$ improvement.

Our attack works due to a special property of the AES S-box. Unlike other ciphers where the S-boxes have no internal structure (and so the 256×256 lookup table looks essentially random), the AES S-box is defined algebraically as the composition of a field inversion in $\mathrm{GF}(2^8)$ and an affine transformation [DR02].[2]

The key insight is that on the output side of the 4-round pre-computed hashtable, a second unknown key byte enters not as a permutation (as the key byte above does) but after a transformation by the AES S-box. By constructing a fingerprint that is invariant under the structured affine action of the AES S-box, we can similarly remove the need to guess this byte, directly reducing the number of keys we need to guess by a factor of 256.

We develop two approaches to removing the need to guess this key byte. One constructs a fingerprint that algebraically cancels out the effect of guessing this key byte, and the other builds a fingerprint by transforming the data into a canonical form that will be the same regardless of the choice of the key byte.

Computational verification. Because these improvements were discovered by a language model rather than by the authors directly, we take particular care to verify correctness through computational means.

First, we prove that the fingerprints we construct are invariant in the way that we require; this ensures that any valid key will still produce a matching fingerprint when looked up in the precomputed table. More challenging is the converse, where we need to show that invalid keys will be rejected with high probability. Here, we both provide a proof formalized in Lean that this statement is true, and also verify the property computationally by exhaustive enumeration over small instances. The computational work also helps us ensure that the actual algorithmic property we need is true in practice.

[2]The S-boxes were designed this way because they give provably strong robustness to differential cryptanalysis and simultaneously are a form of *nothing up my sleeve* proof.

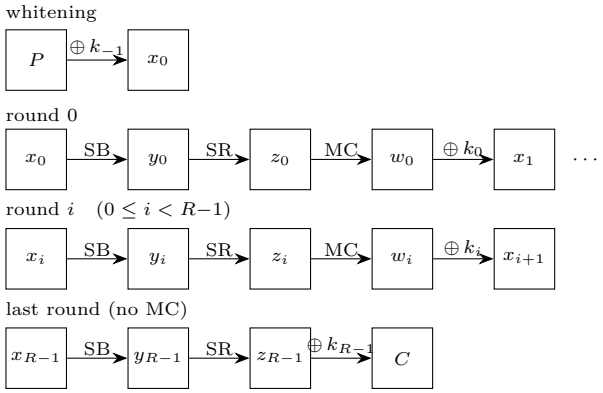

Figure 1: Schematic diagram of AES, with notation labeled.

We additionally verify the correctness of our attack end-to-end on a smaller cipher, AES SR(7,2,2,6) [CMR05], which is identical to AES in every way except we reduce the block size of the cipher from a 4×4 matrix of 8-bit words to a 2×2 matrix of 6-bit words. Demonstrating the attack here is only interesting as a correctness check: brute force requires just $2^{2 \cdot 2 \cdot 6} = 2^{24}$ work, but even constructing the table takes 2^{36} space. Nevertheless it remains a useful validation that all the components of the attack work correctly when composed together.

Finally, we verify that the claimed performance improvements are real by implementing the attack. Because the full attack cannot be run, we implement and measure the two units that the complexity count multiplies out — the cost of one offline table entry and the cost of one online candidate — for both our attack and that of [DFJ13], and we run the complete key recovery on the small-scale cipher above. This lets us check the speedups we claim against measured wall-clock time rather than counted operations alone.

3 Background

3.1 Notation

Figure 1 shows the schematic for R rounds of AES and our (standard) notation. We use k_i to denote round keys starting from $i = -1$ (the first whitening key) and ending with k_{R-1} for the Rth round. An AES round applies four maps to a 4×4 byte state: SB (SubBytes) applies a fixed 8-bit S-box S to each byte; SR (ShiftRows) rotates row r left by r bytes; MC (MixColumns) multiplies each column by a fixed 4×4 MDS matrix over $\mathrm{GF}(2^8)$; and the round key is XORed in. All rounds are identical except the first round is preceded by an extra key XOR, and the last round omits MC.

Within round i the state passes through four named values: x_i (the round input), $y_i = \mathsf{SB}(x_i)$, $z_i = \mathsf{SR}(y_i)$, $w_i = \mathsf{MC}(z_i)$, and then $x_{i+1} = w_i \oplus k_i$; in particular $x_0 = P \oplus k_{-1}$. (The last round omits MC, so $C = z_{R-1} \oplus k_{R-1}$.)

Because MC is linear, $\mathsf{MC}(z_i) \oplus k_i = \mathsf{MC}\big(z_i \oplus \mathsf{MC}^{-1}(k_i)\big)$, so we may add the key in before MC rather than after; we write $u_i := \mathsf{MC}^{-1}(k_i)$ to denote this *equivalent subkey*; for the final round, which omits MC, we set $u_{R-1} := k_{R-1}$.

Bytes of a state s are indexed column-major as $s[0], \ldots, s[15]$: byte j sits at row $j \bmod 4$, column $\lfloor j/4 \rfloor$. When tracing several messages through the cipher, a parenthesized superscript indexes the message: $x_r^{(i)}$ is the round-r state of the ith message.

The *difference distribution table* (DDT) of the S-box is the 256×256 array $\mathrm{DDT}[\Delta][\Gamma] := \#\{x : S(x) \oplus S(x \oplus \Delta) = \Gamma\}$. Entries that are far from zero open the potential to differential cryptanalysis [BS91]; the AES S-boxes [DR02] are designed to mitigate the potential for

such attacks. For the AES S-box every nonzero row sums to 256 with entries in $\{0, 2, 4\}$; in particular, knowing both the input and output difference of a single S-box determines the actual input x up to (on average) one value.

Cost convention. We measure time in AES-encryption units and convert other operations by counting S-box-equivalent lookups, taking one encryption as $C_{\text{AES}} = 160$ such lookups. This is the convention implicit in Derbez et al. [DFJ13, §3.2], who estimate one partial encryption/decryption of a δ-set element (five S-box lookups) as 2^{-5} of an encryption; adopting it makes our complexities directly comparable to their 2^{99}. Memory is measured in 128-bit blocks.

3.2 Demirci–Selçuk meet-in-the-middle

We begin with the meet-in-the-middle attack of Demirci and Selçuk [DS08], which we will demonstrate on 5 and then 7 rounds of AES. This attack is not faster than brute force on AES-128, but will be refined in the following subsections, and so we present it here for illustrative purposes. The attack begins with the following observation: consider the function $f_k : \{0, \dots, 255\} \to \{0, \dots, 255\}$ defined as follows: $f_k(i)$ is obtained by passing a message m with the first byte set to the constant i through 4 rounds of AES, and then taking the first byte of the 4th round, $x_4[0]$. We use this function to calculate the ordered sequence $(f_k(0), \dots, f_k(255))$. That is, to compute f, we first choose some plaintext m at random, and then define the 256 plaintexts $m_i = [i, m[1], m[2], \dots, m[15]]$ where each m_i has the first byte equal to i and the remaining 15 bytes equal to the corresponding byte of m. We then encrypt each of these 256 values $\{m_i\}_{i=0}^{255}$ to obtain $\{c_i\}_{i=0}^{255}$. From each ciphertext in this ordered list we then take the first byte, giving us the ordered list $(c_0[0], c_1[0], \dots, c_{255}[0])$.

We now make the critical observation of [DS08]: while there are 2^{2048} possible sequences in $\{0, 1, ...255\}^{256}$, only a very small fraction of these functions are actually implementable with 4 rounds of AES. As an extreme upper bound, for 256-bit AES, it is easy to see why there are at most $2^{256+15\cdot 8}$ different mappings because there are 2^{256} different keys, and 15 bytes for the plaintext bytes after the first, that define the function. But [DS08] show that f can be parameterized by just 25 bytes of internal state, so there are at most 2^{200} valid sequences. (For now let us take this fact as a given; in future subsections we will show exactly why this is true.) This directly leads to a distinguishing attack: if we enumerate all possible sequences (with 2^{200} work) and insert them into a hashtable, then to distinguish 4-round AES from a random permutation we need only encrypt 256 chosen plaintexts and test whether the resulting sequence is in the table. [3]

Five round cryptanalysis. This directly yields an attack that breaks 5 rounds of AES-256 in 2^{204} space, 256 queries, and $\approx 2^{209}$ time. Each of the 16 tables (one per output byte position) holds 2^{200} sequences of 256 bytes ($= 2^4$ 128-bit blocks) each, i.e. 2^{204} blocks per table; built sequentially, the space stays at 2^{204} blocks. The time is dominated by building these tables: $16 \cdot 2^{200} = 2^{204}$ sequences, each requiring 256 partial encryptions of ≈ 25 S-boxes each, i.e. $\approx 2^{209}$ encryption-equivalents under the cost convention above.

The adversary begins by building the full 2^{200} table by enumerating over the 25 bytes that determine the valid states. Then, they proceed as follows. First, encrypt 256 inputs m_i where the 0th byte $m_i[0] = i$ takes all possible 256 values, and the remaining bytes $m_i[1..15]$ are held constant. Construct the set C by encrypting each m_i giving the ciphertexts $\{c_0, \dots, c_{255}\}$. Then, for each guess $\kappa_{-1}[0] \in [0..255]$ and $\kappa_4[0] \in [0..255]$, the attacker tests whether

$$\left(S^{-1}(c_{\pi(0)}[0] \oplus \kappa_4[0]),\ S^{-1}(c_{\pi(1)}[0] \oplus \kappa_4[0]),\ \dots,\ S^{-1}(c_{\pi(255)}[0] \oplus \kappa_4[0])\right)$$

[3] Clearly if the function were only four rounds we could do this with one encryption, but we will show this trick generalizes to deeper ciphers.

is in the hashtable, where $\pi(i) = i \oplus \kappa_{-1}[0]$ reorders the sequence according to the guessed whitening byte. Exactly one of the 2^{16} assignments to the 16 bits of $(\kappa_{-1}[0], \kappa_4[0])$ will have the resulting 256-tuple as a member of the hashtable, and this happens exactly when the two κ values correspond to the round keys at the associated positions. This directly yields the value of these two key bytes.

We repeat this process 15 times, costing $15 \cdot 2^8$ work, to recover the remaining κ_4 values. First, we build another 2^{200} table where we again vary the first input byte but now measure on the 2nd (or 3rd, and so on) output byte. Then, because $\kappa_{-1}[0]$ is now known, we need only guess over the 2^8 possible values of the byte $k_4[i]$. This yields $8 + 128$ of the 256 bits of the key; the remaining 120 can be brute forced.

Seven round cryptanalysis. Extending from five to seven rounds is not much more challenging. In five rounds, it was necessary to enumerate over the 2 key bytes that determined exactly the data going into the hashtable. Extending from five rounds to seven rounds means we have two more rounds to account for, and we do this by adding one round above and one round below. Thus, our table now is built from round 1 to round 5 (instead of 0 to 4).

On the *input* side, naïvely one might expect to guess the entire whitening key, but because the δ-set is defined by a single byte of x_1, only the four bytes $k_{-1}[0, 5, 10, 15]$ that feed that byte through $\mathsf{MC} \circ \mathsf{SR} \circ \mathsf{SB}$ are needed — a 2^{32} guess.

On the *output* side, we need $x_5[0]$ from the ciphertext. Peeling the last round at the four anti-diagonal positions $u_6[0, 7, 10, 13]$ (another 2^{32} guess) yields column 0 of x_6. One round further back, naïvely we would need all of $k_5[0, 1, 2, 3]$ since $z_5 = \mathsf{MC}^{-1}(x_6 \oplus k_5)$ mixes the whole column, but the equivalent-subkey trick collapses it:

$$z_5 \;=\; \mathsf{MC}^{-1}(x_6) \oplus \underbrace{\mathsf{MC}^{-1}(k_5)}_{=:\,u_5} \;\Longrightarrow\; z_5[0] \;=\; \mathsf{MC}^{-1}(x_6)[0] \oplus u_5[0],$$

so the single byte $u_5[0] = 14 \cdot k_5[0] \oplus 11 \cdot k_5[1] \oplus 13 \cdot k_5[2] \oplus 9 \cdot k_5[3]$ suffices — a fixed linear combination of $k_5[\text{col } 0]$, not any individual k_5 byte.

The 7-round attack therefore guesses $4 + 4 + 1 = 9$ key bytes (2^{72}) on top of the 2^{200} precomputation; well above exhaustive search for AES-128, but the structure is the one all subsequent attacks refine.

3.3 Dunkelman–Keller–Shamir: multisets and differential enumeration

Dunkelman, Keller, and Shamir [DKS10] improve this attack with two ideas: a *multiset* tabulation that absorbs one guessed key byte, and a *differential enumeration* that cuts the table from 2^{192} entries to 2^{127}.

Multisets. Instead of the ordered 256-tuple, [DKS10] store the unordered *multiset* of differences $\{\, x_5[0]^{(i)} \oplus x_5[0]^{(0)} : i = 0, \ldots, 255 \,\}$. They show (their Obs. 4) that this multiset is determined by 24 explicit bytes of internal state — the full 16-byte $x_3^{(0)}$, four bytes $x_2^{(0)}[0..3]$, and four bytes $k_3[0, 5, 10, 15]$ — and so can take at most 2^{192} values. Moreover, each δ-set is counted 2^8 times in this bound (once for each choice of the reference message $P^{(0)}$ among its members), and fixing that choice canonically leaves 2^{184} possible multisets. But the primary objective of moving to multisets is that they remove the dependence on the key byte $k_0[0]$: the byte $k_0[0]$ enters the attack only by determining which plaintext corresponds to which index i, and a quantity that ignores the order is independent of it.

Differential enumeration. The expensive parameter above is the full 16-byte $x_3^{(0)}$. [DKS10] observe that if the reference message $P^{(0)}$ is one of the two messages (m, m') of a *right pair* for a particular truncated differential, then $x_3^{(0)}$ is pinned to one of only 2^{64} key-independent values.

The differential is a four-round 1→4→16→4→1 pattern over rounds 1–4: Δx_1 is nonzero only in byte 0, and Δx_5 is nonzero only in byte 0. Round 3's SubBytes sits at the "16" in the middle, and the argument is a meet-in-the-middle on its input and output differences:

- *Forward.* From Δx_1 (one byte), one round gives Δx_2 nonzero only in column 0; after round-2 SB, Δy_2 is still four bytes in column 0 (and so takes at most 2^{32} values) and the linear layer of round 2 carries this to Δx_3, fully active but taking at most 2^{32} possible values.
- *Backward.* From Δx_5 (one byte), one inverse round gives Δx_4 nonzero only on the diagonal $\{0, 5, 10, 15\}$ (again, taking at most 2^{32} values). The inverse linear layer of round 3 carries this to Δy_3, again fully active but with at most 2^{32} possible values.
- *Meet.* For each of the $2^{32} \times 2^{32} = 2^{64}$ choices of $(\Delta x_3, \Delta y_3)$, the DDT at each of the 16 byte positions fixes $x_3^{(0)}[j]$ up to (on average) one value. Hence $x_3^{(0)}$ lies in a set of at most 2^{64} values that can be enumerated offline without knowing the key.

Replacing the 2^{128} free choices of $x_3^{(0)}$ by these 2^{64} reduces the 2^{184} multisets by a factor of only 2^{57}, not 2^{64}: the 2^8 saving that brought 2^{192} down to 2^{184} came from the freedom to pick *any* δ-set member as $P^{(0)}$, but the differential now forces $P^{(0)}$ to be one of the two right-pair members, so only a factor of 2 of that saving survives. The net result is that this attack requires storing $2^{184} \cdot 2^{-64} \cdot 2^7 = 2^{127}$ multisets ($\approx 2^{129}$ blocks of memory).

Finding the right pair. The differential spans rounds 1–4, but the attacker controls only round 0 and observes only round 6. The attack addresses this limitation statistically.

On the output side, if the quantity Δx_5 is nonzero in one byte then ΔC must be nonzero in exactly the four anti-diagonal bytes $\{0, 7, 10, 13\}$. This makes the differential detectable from only input-output behavior.

On the input side, the attacker constructs a *structure* of 2^{32} plaintexts taking all values on the diagonal $\{0, 5, 10, 15\}$. After one round of the cipher, the resulting structure is guaranteed to contain at least one copy of every possible Δx_1 restricted to byte 0. (It also contains many other Δx_1 values that are spurious, but these will probably not have the correct output differential.) A structure yields $\approx 2^{63}$ pairs that can be compared. On average 2^{-33} of them pass the ciphertext filter ($63 - 96 = -33$). Moreover, a filter-passing pair is a *right* pair with probability $\approx 2^{-48}$ (because the two 4→1 MC collapses, three zero bytes each). Hence 2^{81} structures ($D = 2^{113}$ chosen plaintexts) yield $\approx 2^{48}$ candidate pairs, of which ≈ 1 is right.

We then look up each of these (m, m') candidate pairs in the table that we computed offline as we did in the prior attack. Each of the wrong pairs will fail to match when looking up in the hashtable, but the one right pair will succeed—and when it does so, allows the attacker to directly compute the value of the key using the same techniques as in the prior section.

Result. The basic 7-round attack of [DKS10] (their §6) has $D = 2^{113}$, $T \approx 2^{113}$, $M = 2^{129}$; a time/memory tradeoff with parameter n gives $D = T = 2^{103+n}$, $M = 2^{129-n}$, equalized at 2^{116} for $n = 13$.

3.4 Derbez–Fouque–Jean: efficient tabulation

Derbez, Fouque, and Jean [DFJ13] make three changes: first, they tighten the analysis from [DKS10], second, they show that several key bytes are constrained and are not completely free, and third they introduce a new differential attack that trades off time for queries, bringing the 7-round attack below 2^{100}. Our attack is independent of the proof technique

from this paper, and so we just state the property and sketch the intuition; we refer the reader to [DFJ13] for the full rebound argument.

Tighter table (Prop. 2 of [DFJ13]). The attack from [DKS10] used the right-pair differential only to pin the 16-byte middle state $x_3^{(0)}$. The improvement here relies on the observation that the *same* differential also constrains the remaining eight parameter bytes via a rebound argument. Specifically, the following ten bytes

$$\Delta z_1[0], \quad x_2^{(0)}[0..3], \quad \Delta w_4[0], \quad z_4^{(0)}[0..3]$$

suffice to determine *all* 24 of the parameters in [DKS10].

Propagating $\big(\Delta z_1[0],\, x_2^{(0)}[0..3]\big)$ forward and $\big(\Delta w_4[0],\, z_4^{(0)}[0..3]\big)$ backward fixes the values of $(\Delta x_3, \Delta y_3)$ exactly, the DDT then fixes $x_3^{(0)}$, and $z_4^{(0)}[0..3]$ gives $x_4^{(0)}[\text{diag}]$ via one SB^{-1}. The table thus holds 2^{80} multisets, $M \approx 2^{82}$ blocks.

Online key guesses. For each candidate pair, [DFJ13] observe that the four bytes $k_{-1}[0, 5, 10, 15]$ are not completely free variables. Because $\Delta w_0[1, 2, 3] = 0$ (the input-side 4→1 collapse) is a 24-bit filter, there are only $\approx 2^8$ candidate keys $k_{-1}[0, 5, 10, 15]$ that are valid. Symmetrically, $u_6[0, 7, 10, 13]$ are likewise reduced to $\approx 2^8$ candidates through the differential $\Delta z_5[1, 2, 3] = 0$. The byte $u_5[0]$ remains unconstrained and must still be guessed via brute force. This brings the per-pair inner-loop to $2^8 \cdot 2^8 \cdot 2^8 = 2^{24}$ multiset constructions and lookups.

Improved differential attack. With 2^{48} candidate pairs from the attack above, the lookup phase costs $2^{48} \cdot 2^{24} \cdot 256 \cdot 2^{-5} = 2^{75}$ time and 2^{113} encryptions. [DFJ13] show it is possible to widen the differential to *two* active bytes at z_2 and thus reduce the data complexity to 2^{105} at a cost of increasing the time complexity by 2^{24}. Instead of structuring the differential in the first round so one diagonal takes all possible values, structure the first round so that we vary *two* diagonals (2^{64} plaintexts) yielding 2^{127} pairs each. The same 96-bit ciphertext filter and a 72-bit middle constraint leave 2^{72} candidate pairs from 2^{41} structures, i.e. $D = 2^{105}$. The cost is one extra parameter byte (the second Δz_2 value), so the table grows to 2^{88} multisets, $M \approx 2^{90}$, and similarly the time complexity increases to $2^{72} \cdot 2^{24} \cdot 256 \cdot 2^{-5} = 2^{99}$.

Subsequent work. Derbez and Fouque [DF13] automated the search over this family of attacks and extended it to more rounds of AES-192/256, and Bar-On et al. [BODK+20] pushed the practical-data and low-memory frontier for reduced-round AES-128; neither improves the 2^{99} time at $D{=}2^{105}$ for 7-round AES-128, which is the number this paper improves. Followup work has extended attacks like this to more rounds [LJW14, LJ16].

4 Our attack

We improve on this line of work. Section 4.1 introduces our core conceptual contribution: an algebraic *bridge* across the round-5 S-box that eliminates the requirement that the attacker guess the byte $u_5[0]$, reducing the inner loop from 2^{24} to 2^{16} per pair. This is the output-side analogue of the multiset trick of [DKS10] (which eliminated the input-side byte $k_0[0]$). Section 4.2 introduces an improved variant of this approach that is both faster and also more accurate.

4.1 Polynomial Ratios

Let us first recall which key bytes must be guessed per candidate pair. At the top of the cipher, we need to correctly guess the first 4 key bytes $k_{-1}[0, 5, 10, 15]$ (but these only require guessing 8 bits because of constraints). We do not need to guess any bytes from the subsequent round because the multiset is invariant to the choice of the key-byte $k_0[0]$. At the bottom of the cipher, we need to correctly guess the last 4 key bytes $u_6[0, 7, 10, 13]$ (but again these are determined by 8 bits). But we *do* need to guess the one byte $u_5[0]$ of the second-to-last round, because the S-box between it and the match point prevents absorbing it into the table. The inner loop is thus $2^8 \cdot 2^8 \cdot 2^8 = 2^{24}$ lookups per pair; our goal is to remove the $u_5[0]$ factor.

Specifically, for the set of 256 plaintexts, suppose that we correctly have guessed the key bytes on the top of the cipher. That is, we have a set of 256 plaintexts $\{m_i\}_{i=0}^{255}$ with the property that $\{x_1[0]\}$ takes all 256 values and $x_1[1..15]$ are equal to a fixed constant. Encrypting through 7 rounds yields 256 ciphertexts $\{C^{(\omega)}\}$. With the bottom-side guess $u_6[0, 7, 10, 13]$ also correct, we peel round 6 on those four positions and obtain $x_6[0, 1, 2, 3]$ (column 0 of x_6) for every ciphertext. But now we are stuck: we don't know $u_5[0]$. Previously, we brute forced all 256 values of $u_5[0]$ (the equivalent subkey byte) which allowed us to learn the value of $x_5[0]$. We will now proceed without this brute force.

Since $x_6 = w_5 \oplus k_5$, we have $w_5 = x_6 \oplus k_5$, and pushing backward through MC gives

$$\begin{aligned} z_5 \;=\; \mathsf{MC}^{-1}(w_5) \;&=\; \mathsf{MC}^{-1}(x_6 \oplus k_5) \\ &=\; \mathsf{MC}^{-1}(x_6) \;\oplus\; \mathsf{MC}^{-1}(k_5) \\ &=\; \mathsf{MC}^{-1}(x_6) \;\oplus\; u_5. \end{aligned}$$

We can continue: since SR^{-1} is linear,

$$y_5 \;=\; \mathsf{SR}^{-1}(z_5) \;=\; \mathsf{SR}^{-1}\big(\mathsf{MC}^{-1}(x_6)\big) \;\oplus\; \mathsf{SR}^{-1}(u_5).$$

For byte 0 this simplifies further: row 0 is fixed by SR, so $y_5[0] = z_5[0]$ and $\mathsf{SR}^{-1}(u_5)[0] = u_5[0]$. Hence

$$x_5[0] \;=\; S^{-1}\big(y_5[0]\big) \;=\; S^{-1}\Big(\mathsf{MC}^{-1}(x_6)[0] \;\oplus\; u_5[0]\Big),$$

which depends on k_5 only through the single byte $u_5[0]$. Let $v := \mathsf{MC}^{-1}(x_6)[0]$, which we can compute from the known column $x_6[0..3]$, and $\kappa := u_5[0]$. Across the 256 ciphertexts of the δ-set we thus obtain 256 known bytes $\{v_0, \ldots, v_{255}\}$, and the values we would like to tabulate are $S^{-1}(v_\omega \oplus \kappa)$. In order to be able to use a precomputed lookup table, it is necessary that the queries we are making do not depend on any key bytes; the dependence of these values on κ is exactly where prior attacks got stuck.

Observation: AES S-box structure. The AES S-box is defined as $S(t) = L(t^{-1}) \oplus 63_{16}$, where t^{-1} is the field inversion in GF(2^8) (with $0^{-1} := 0$), L is a fixed GF(2)-linear bijection, and 63_{16} is the base-16 constant 99. Throughout, field arithmetic is in GF(2^8) reduced by the AES polynomial $x^8{+}x^4{+}x^3{+}x{+}1$. [4]

Write $a_\omega := x_5[0]^{(\omega)}$ for the ω-th member of the δ-set and fix the reference value $a := a_0$. Offline, it is easy to calculate the differences because the 11 parameters of [DFJ13] determine every difference $\Delta x_5[0]$ (though not the absolute values), so $d_\omega := a \oplus a_\omega$ (indeed, this is exactly the $\Delta x_5[0]$ multiset of [DFJ13]). However, online the attacker only knows

[4] Throughout this paper we write x^{-1} for the AES inversion for readability; other work writes x^{254} which handles the $x{=}0$ case directly. When the $0^{-1} := 0$ case is important, we discuss it explicitly where it matters (e.g., §4.1).

the value of $v_\omega = S(a_\omega) \oplus \kappa$, where κ is the unknown key byte $\kappa = u_5[0]$. By computing the delta $v_0 \oplus v_\omega$ it is possible for us to cancel out the constants via

$$v_0 \oplus v_\omega \;=\; (S(a) \oplus \kappa) \oplus (S(a_\omega) \oplus \kappa) \;=\; S(a) \oplus S(a_\omega),$$

and so the only remaining unknown is the scalar $s := a$, the value of the reference message at round 5, which neither the online nor the offline phase knows.[5] However, if we perform the computation

$$\begin{aligned}
&v_0 \oplus v_\omega \\
&\quad = \big[S(a) \oplus \kappa\big] \oplus \big[S(a_\omega) \oplus \kappa\big] && \text{(def. of } v\text{)} \\
&\quad = S(a) \oplus S(a_\omega) && (\kappa \oplus \kappa = 0) \\
&\quad = \big[L(a^{-1}) \oplus 63_{16}\big] \oplus \big[L(a_\omega^{-1}) \oplus 63_{16}\big] && \text{(def. of } S\text{)} \\
&\quad = L(a^{-1}) \oplus L(a_\omega^{-1}) && (63_{16} \oplus 63_{16} = 0) \\
&\quad = L\big(a^{-1} \oplus a_\omega^{-1}\big) && (L \text{ linear}) \\
&\Longrightarrow\; L^{-1}(v_0 \oplus v_\omega) \\
&\quad = a^{-1} \oplus a_\omega^{-1} && \text{(apply } L^{-1}\text{)} \\
&\quad = \frac{a \oplus a_\omega}{a\, a_\omega} && \text{(common denom.)} \\
&\quad = \frac{d_\omega}{s\,(s \oplus d_\omega)} && (s{:=}a,\ d_\omega{:=}a{\oplus}a_\omega) \\
&\quad = \frac{d_\omega}{s^2 \oplus s\, d_\omega} && \text{(distribute)}
\end{aligned}$$

and then taking the reciprocal of both sides and defining

$$g_\omega \;:=\; \big(L^{-1}(v_0 \oplus v_\omega)\big)^{-1}, \qquad \omega = 1, \ldots, 255,$$

we obtain the key identity

$$g_\omega \;=\; s^2 \cdot d_\omega^{-1} \;\oplus\; s.$$

In this derivation, the left-hand side g_ω can be computed online from the v_ω. The right-hand side involves the offline-known d_ω—but sadly also contains an unknown byte s. Removing the dependence on this byte s is our goal, and requires some work.

The byte s enters $g_\omega = s^2 \cdot d_\omega^{-1} \oplus s$ twice: once as an additive shift s and once as a multiplicative scale s^2. We remove these in turn—the shift by a second XOR-difference, and the scale by then taking a ratio—leaving a quantity that both sides can compute independently and compare.

Removing the additive shift. Take any two indices $\omega \neq \omega'$ in $\{1, \ldots, 255\}$ and XOR the corresponding g-values:

$$\begin{aligned}
h_{\omega,\omega'} \;&:=\; g_\omega \oplus g_{\omega'} \\
&= \big(s^2 \cdot d_\omega^{-1} \oplus s\big) \oplus \big(s^2 \cdot d_{\omega'}^{-1} \oplus s\big) && \text{(key identity, twice)} \\
&= s^2 \cdot d_\omega^{-1} \oplus s^2 \cdot d_{\omega'}^{-1} && (s \oplus s = 0) \\
&= s^2 \cdot \big(d_\omega^{-1} \oplus d_{\omega'}^{-1}\big) && \text{(factor } s^2\text{)}.
\end{aligned}$$

[5] Note that this bridge relation fails when $a_\omega \in \{0, s\}$ — what we call "bad indices". We address these cases separately in §4.1 and Appendix B.

The same reference index 0 is used in both g_ω and $g_{\omega'}$, so the same s appears in both and the additive copy cancels. Online, the attacker computes the left-hand side as $h_{\omega,\omega'} = \big(L^{-1}(v_0 \oplus v_\omega)\big)^{-1} \oplus \big(L^{-1}(v_0 \oplus v_{\omega'})\big)^{-1}$ which is entirely computable from quantities the attacker knows. Offline, the attacker knows $d_\omega^{-1} \oplus d_{\omega'}^{-1}$, with the only remaining unknown being the multiplicative scale s^2, and it is the *same* scale for every pair (ω, ω').

A naïve ratio (that doesn't work). Since the same scale s^2 multiplies every $h_{\omega,\omega'}$, a naïve idea would be to divide one by another. If we pick any four indices $\omega, \omega', \mu, \mu' \in \{1, \dots, 255\}$ then we can obtain the quantity

$$\frac{h_{\omega,\omega'}}{h_{\mu,\mu'}} \;=\; \frac{s^2\big(d_\omega^{-1} \oplus d_{\omega'}^{-1}\big)}{s^2\big(d_\mu^{-1} \oplus d_{\mu'}^{-1}\big)} \;=\; \frac{d_\omega^{-1} \oplus d_{\omega'}^{-1}}{d_\mu^{-1} \oplus d_{\mu'}^{-1}},$$

which is free of s. Online, the attacker can evaluate the left side from its g's. And offline, we can evaluate the right side from its d's, so we might hope to build a table out of these ratios.

This fails for the same reason that prior work [DKS10] needed to resort to multisets: the two sides do not share an indexing of the δ-set. The offline ω enumeration of the 255 differences produced by the table parameters will not match the online ω enumeration produced by computing ciphertexts. Any single ratio depends on *which* four indices were picked, and there is no quadruple the two sides can agree on in advance.

One could restore order-invariance by brute force: both sides compute the ratio for *every* choice of $(\{\omega, \omega'\}, \{\mu, \mu'\})$ and hash the resulting multiset. Both sides would then agree. But there are $\frac{1}{2}\binom{255}{2}^2 \approx 2^{29}$ such quadruples, and so evaluating the fingerprint costs $\sim 2^{29}$ field operations per lookup—too slow to be of practical use.

Power-sum invariants. We need a fingerprint that is symmetric in the indices (so both sides can compute it without agreeing on an ordering) yet still carries enough information to serve as a fingerprint and is possible to calculate efficiently. The most naïve symmetric fingerprint is the XOR-sum because it is order-invariant

$$\bigoplus_{\omega<\omega'} h_{\omega,\omega'}.$$

Sadly this quantity is identically zero: each g_ω appears in an even number (254) of pairs, so everything cancels in characteristic 2:

$$\begin{aligned}
\bigoplus_{1\le\omega<\omega'\le 255} h_{\omega,\omega'} &= \bigoplus_{\omega<\omega'} \big(g_\omega \oplus g_{\omega'}\big) && \text{(def. of } h\text{)} \\
&= \bigoplus_{\omega=1}^{255} \Big(\underbrace{g_\omega \oplus \cdots \oplus g_\omega}_{254 \text{ copies}}\Big) && \text{(re-ordering)} \\
&= \bigoplus_{\omega=1}^{255} 0 && (x \oplus x = 0) \\
&= 0.
\end{aligned}$$

But raising to a power before summing breaks this parity. For some $m \ge 1$ define

$$P_m \;:=\; \bigoplus_{1\le\omega<\omega'\le 255} h_{\omega,\omega'}^m \;=\; s^{2m} \cdot \underbrace{\bigoplus_{\omega<\omega'} \big(d_\omega^{-1} \oplus d_{\omega'}^{-1}\big)^m}_{=:\, Q_m},$$

so that $P_m = s^{2m} Q_m$.

This quantity *still* has a dependence on the value s, but we can finally address this with a quotient (as we did above). If we calculate

$$I_{m,n} := \frac{P_m^n}{P_n^m} = \frac{(s^{2m}Q_m)^n}{(s^{2n}Q_n)^m} = \frac{s^{2mn}}{s^{2nm}} \cdot \frac{Q_m^n}{Q_n^m} = \frac{Q_m^n}{Q_n^m},$$

then finally this is a quantity independent of s. Thus the online phase (from $\{g_\omega\}$, via P_m) and the offline phase (from $\{d_\omega\}$, via Q_m) compute the same value $I_{m,n}$. Simultaneously, it also allows us to form many $I_{m,n}$ outputs that allow us to fill the table so that we can obtain enough independent bytes to form a ≥ 96-bit fingerprint.

We now address a number of details.

Reference agreement. The above fingerprint assumes that both the offline and online phases use the same reference message. Because $d_\omega = x_5[0]^{(\omega)} \oplus x_5[0]^{(0)}$, choosing a different message as "ω=0" shifts every d_ω by a constant; inversion does not commute with that shift, so $\{d_\omega^{-1}\}$ — and hence $I_{m,n}$ — changes. The two phases must therefore agree on the reference, yet they cannot communicate.

Agreement is nonetheless automatic. Recall that each table entry is indexed by eleven parameter bytes (§3.4), of which eight are *absolute* state values: $x_2[0..3]$ and $z_4[0..3]$ of one particular message. That message — whichever plaintext, under the secret key, actually produces those eight bytes at rounds 2 and 4 — is the entry's *anchor*. When the offline phase computes the entry's 255 differences d_ω, it does so relative to the anchor (the d-values are "how does $x_5[0]$ change as we vary $x_1[0]$ away from the anchor"), so the anchor is the entry's built-in reference.

Online, the attacker builds the δ-set around the candidate-pair member P and computes $\{g_\omega\}$ relative to P; so P is online's reference. Among the 2^{88} table entries, exactly one has parameter bytes $x_2[0..3], z_4[0..3]$ equal to P's *actual* round-2 and round-4 state under the secret key — and that entry, by definition, is anchored at P. Any other entry is anchored at some other message and produces a different $I_{m,n}$. So the single entry that can match already has the same reference as the online side; no search over references is needed.[6]

The lookup itself is an $O(1)$ hash-table probe: the offline table is keyed by the 12-byte $I_{m,n}$ fingerprint, and online computes its fingerprint once and queries.

Choosing the exponents. The exponents must be chosen so that the resulting $I_{m,n}$ bytes are non-degenerate and pairwise independent. We require $\gcd(m, 255) = 1$ so that $t \mapsto t^m$ is a bijection on $\mathrm{GF}(2^8)^\times$, and we take at most one exponent from each Frobenius coset $\{m, 2m, 4m, \dots\}$ mod 255, since $P_{2m} = P_m^2$ forces $I_{m,2n} = I_{m,n}^2$. There are exactly 15 such coset representatives coprime to 255:

$$\mathcal{E} = (7, 11, 13, 19, 23, 29, 31, 37, 43, 47, 53, 59, 61, 91, 127).$$

Since the unknown s removes one degree of freedom, at most 14 algebraically independent $I_{m,n}$ invariants exist. A further subtlety arises from the fact that the ratio $I_{m_0,n} = P_{m_0}^n / P_n^{m_0}$ collapses to 0 whenever $P_{m_0} = 0$ and is undefined whenever $P_n = 0$. Left unguarded, the $\approx 2^{-8}$ fraction of entries with $P_{m_0} = 0$ would all share the fingerprint $\mathbf{0}$. We handle this by defining the fingerprint via the first thirteen *nonzero* P_m along $\mathcal{E}$: m_0 is the first index with $P_{m_0} \neq 0$ and $n_1, \dots, n_{12}$ the next twelve. Because $P_m = s^{2m} Q_m$ with $s \neq 0$, online and offline see the same zero pattern and select the same exponents, so no right pair is lost to a one-sided skip. Fewer than 13 of the 15 sums are nonzero with probability $\leq \binom{15}{3} 2^{-24} \approx 2^{-15}$, which we ignore. Every I_{m_0,n_j} then lies in $\mathrm{GF}(256)^\times$; we

[6] One could instead try all 256 δ-set members as reference and look each up; this works but costs a factor 256, exactly undoing the saving of this section.

verify empirically (§5.1) that the resulting 12-byte tuple behaves consistently with the ≈ 96-bit structural maximum.

The 0^{-1} corner case. The identity $g_\omega = s^2 d_\omega^{-1} \oplus s$ assumes that we are operating over proper field inverses, but the AES S-box sets $0^{-1} := 0$ arbitrarily. This means that when $a_\omega \in \{0, s\}$, the online evaluation of g_ω yields one of $\{0, s\}$ while the formula gives the other. Call these the *bad* indices and let b be their count (on expectation, ≈ 2 indices will be bad under the standard randomness heuristic). Because the bad slots hold only the two values 0 and s, and P_m is symmetric, the discrepancy depends only on the parity of b: $P_m^{\text{online}} \oplus s^{2m} Q_m = (b \bmod 2) \cdot D_m$, where D_m depends on the remaining g_ω but not on *which* indices are bad. The fingerprints therefore agree whenever b is even, but when b is odd (roughly half the time), the online and offline fingerprints will disagree, and the correct table entry will be missed.

To cover odd b, both sides also compute the fingerprint after appending one synthetic zero (online appends $g = 0$ and offline appends $d^{-1} = 0$), which increments b and flips the parity. Online looks up both fingerprints and accepts a hit if either is present; exactly one of the two will match, except when $s = 0$ (which occurs with probability 2^{-8}). We accept this failure, and absorb it as a 256/255 factor on the data requirement. [7] This doubles the table entries ($2^{88} \to 2^{89}$) but not the computation: the second fingerprint reuses the accumulated p_r (§A.1) and costs only the ≈ 60-lookup combine step.

The exponent selection of §4.1 needs one amendment for this second fingerprint. Appending the synthetic zero makes the multiset even-sized (256 elements rather than 255), and at even size every Lucas term of a weight-3 exponent carries a factor of the single-index sum $S_1 = \bigoplus_\omega g_\omega$ (§A.1). The five weight-3 exponents $7, 11, 13, 19, 37$ therefore vanish *together* whenever $S_1 = 0$ (probability 2^{-8}), leaving only ten nonzero sums, so the appended fingerprint is undefined (measured rate $2^{-8.0}$). Since this second fingerprint is needed only when b is odd (about half of right pairs), the loss could simply be absorbed as a further 2^{-9} factor on the data requirement. We will not need to pay even this much: the stronger fingerprint we develop in §4.2 removes the requirement entirely, handling $S_1{=}0$ by an explicit fallback (Appendix B) rather than by giving up on the entry, and it is the one our final complexity uses.

Can we do better? The bridge developed in this section eliminates one guessed byte. Two natural questions are whether the same trick can eliminate a *second* byte, and whether the fingerprint can be computed from *fewer* than 255 values. Both answers are no.

No second bridgeable byte. The bridge works because $u_5[0]$ is the last unknown byte between the ciphertext and the match point $x_5[0]$, with only a single S-box in between; the S-box's invert-then-affine structure turns that one unknown into a single $\mathrm{AGL}(1, 256)$ action on the data, which the fingerprint is designed to ignore. Every other byte in the attack (the eight remaining online key bytes and the eleven offline parameters) reaches $x_5[0]$ through at least one MixColumns, which mixes four bytes into four and destroys the clean group action.

All 255 values are needed. One might also hope to compute the fingerprint from a small subset of the 255 δ-set values. This fails because online and offline index the δ-set differently: the bijection between their indexings is $\sigma_\xi(v) = S(\xi \oplus v) \oplus S(\xi)$ where $\xi = x_1^{(0)}[0]$ depends on the unguessed key byte $k_0[0]$. As ξ ranges over all possibilities, the permutations σ_ξ generate the full symmetric group $\mathrm{Sym}(255)$,[8] so any fingerprint that

[7] We note for clarity that s is a purely online quantity; the offline table is built from the differences d_ω alone and has no s-dependent branch.

[8] It can be shown that σ_0 has cycle type $(166, 81, 7, 1)$, so σ_0^{13446} is a 7-cycle; that $\langle \sigma_\xi \rangle$ is 2-transitive; and that every σ_ξ is an odd permutation. Jordan's theorem then gives $\langle \sigma_\xi \rangle \supseteq \mathrm{Alt}(255)$, and the odd generators force equality with $\mathrm{Sym}(255)$.

Algorithm 1 ConstructTable(payload) — an entry's 255-element difference sequence from its anchor bytes. No key byte appears.

0. $(x_2[0..3],\ x_3[0..15],\ x_4[0,5,10,15]) \leftarrow$ Anchors(payload) (DDT solve of §3.4 at the payload's branch bits)
1. **for** $e = 1, \ldots, 255$ **do**
2. $\quad \Delta x_2 \leftarrow (2e,\ e,\ e,\ 3e)$ on column 0, 0 elsewhere
3. $\quad \Delta y_2[j] \leftarrow S(x_2[j]) \oplus S(x_2[j] \oplus \Delta x_2[j])$, $j \in$ col 0
4. $\quad \Delta x_3 \leftarrow \mathsf{MC}(\mathsf{SR}(\Delta y_2))$; $\Delta y_3[j] \leftarrow S(x_3[j]) \oplus S(x_3[j] \oplus \Delta x_3[j])$, all j
5. $\quad \Delta x_4 \leftarrow \mathsf{MC}(\mathsf{SR}(\Delta y_3))$; $\Delta y_4[j] \leftarrow S(x_4[j]) \oplus S(x_4[j] \oplus \Delta x_4[j])$, $j \in \{0,5,10,15\}$
6. $\quad d_e \leftarrow 2\Delta y_4[0] \oplus 3\Delta y_4[5] \oplus \Delta y_4[10] \oplus \Delta y_4[15]$ (row 0 of MC; $= \Delta x_5[0]$)
7. **return** $(d_1, \ldots, d_{255})$

works for all ξ must be symmetric in all 255 arguments and cannot depend on a proper subset.

Accounting. We adopt the efficient differential of [DFJ13, §3.3], which requires 2^{41} structures of 2^{64} chosen plaintexts, and so we require 2^{105} data in total. Each structure yields $\approx 2^{127}$ pairs, and the 96-bit ciphertext filter leaves $2^{127+41-96} \approx 2^{72}$ candidate pairs across all structures. For each candidate we enumerate 2^8 values of $k_{-1}[0,5,10,15]$ and 2^8 values of $u_6[0,7,10,13]$; we no longer need to enumerate $u_5[0]$. And so finally we need to perform $2^{72} \cdot 2^{16} = 2^{88}$ fingerprint computations. Converting this count to encryption units requires the per-entry cost developed in Appendix A.1–A.2; the result is summarized in §4.1.1.

The precomputed table has 2^{88} entries (the 11-byte parameter space of [DFJ13, §3.3]). Each entry stores its 12-byte fingerprint as the lookup key, and the 11 parameter bytes plus ≈20 DDT branch bits (≈26 bytes total) as a payload. We write ConstructTable(payload) for the offline per-entry procedure that produced the entry (Algorithm 1): the payload's branch bits select DDT roots and so fix the entry's *anchor* state bytes $\left(x_2^{(0)}[0..3],\ x_3^{(0)},\ x_4^{(0)}[0,5,10,15]\right)$ (§3.3–§3.4); from these, for each $e = \Delta z_1[0] \in \{1, \ldots, 255\}$ it re-traces the δ-difference through the S-box layers of rounds 2–4 and returns the entry's difference sequence $\{d_e = \Delta x_5[0]_e\}$, at 4+16+4 = 24 S-box lookups per index ($\approx 2^{11}$ amortized over the Gray-code walk of Appendix A.2).

The payload is what lets the online phase act on a hit: it is needed both to reject false positives (below) and to complete the key (Algorithm 3).

False positives. A 12-byte fingerprint is a ≈96-bit filter, so a wrong guess collides with some table entry with probability $\approx 2^{88}/2^{96} = 2^{-8}$. Over 2^{88} lookups that is $\approx 2^{80}$ spurious hits. Fortunately, it is not that hard to efficiently reject these false positives.

Each hit returns the payload stored with the entry, and one pass of ConstructTable over that payload rebuilds the entry's full 255-element sequence $\{d_\eta\}$ at a cost of $\approx 2^{11}$ lookups. This puts both sides of the bridge in hand at once: online holds $\{g_\omega\}$ and its power sums P_m, while the payload gives $\{d_\eta\}$ and its power sums Q_m. If the hit is genuine the two are related by the bridge identity $P_m = s^{2m} Q_m$, so the test is to recover the s that this identity implies and then check that it really does carry one side onto the other.

Recovering s. Use the sums belonging to the parity that produced the hit: set $(\hat{P}_m, \hat{Q}_m) := (P_m, Q_m)$ for a raw hit, and $(\hat{P}_m, \hat{Q}_m) := (P_m \oplus p_m,\ Q_m \oplus q_m)$ — the appended-parity sums of §A.1 — for an Add-0 hit. Because $s \neq 0$, a genuine hit has $\hat{P}_m = 0$ exactly when $\hat{Q}_m = 0$. We therefore take the first exponent m_0 of the chain (§4.1) with $\hat{P}_{m_0} \neq 0$, reject the hit at once if $\hat{Q}_{m_0} = 0$, and otherwise set $s = (\hat{P}_{m_0}/\hat{Q}_{m_0})^{1/(2m_0)}$ — a root that exists because $\gcd(2m_0, 255) = 1$ for every exponent in the chain.

Comparing the two sides. With s known, the offline sequence can be pushed forward to the online values it predicts, $d_\eta \mapsto s^2 d_\eta^{-1} \oplus s$. These do not agree as *full* multisets: at the bad indices of §4.1 the online value and the affine image swap $0 \leftrightarrow s$, so the two sides

Algorithm 2 BridgeConsistency(payload; $\{g_\omega\}, P_m$) — reject a false-positive table hit.

1. $\{d_\eta\} \leftarrow$ ConstructTable(payload); $q_m \leftarrow \bigoplus_\eta d_\eta^{-m}$; Q_m per (3)
2. $(\hat{P}_m, \hat{Q}_m) \leftarrow (P_m, Q_m)$ if the hit was raw, $(P_m \oplus p_m,\ Q_m \oplus q_m)$ if Add-0
3. $m_0 \leftarrow$ first exponent in the chain (§4.1) with $\hat{P}_{m_0} \neq 0$; reject if $\hat{Q}_{m_0} = 0$; $s \leftarrow (\hat{P}_{m_0}/\hat{Q}_{m_0})^{1/(2m_0)}$
4. accept iff $\{g_\omega\}_\omega \setminus \{0, s\} = \{s^2 d_\eta^{-1} \oplus s\}_\eta \setminus \{0, s\}$ as multisets

match only when their bad slots happen to balance — for the true hit, about half the time. Deleting those two values from both sides removes the discrepancy, and we accept the hit if and only if

$$\{g_\omega\} \setminus \{0, s\} \;=\; \{s^2 d_\eta^{-1} \oplus s\} \setminus \{0, s\}$$

as multisets, where $\setminus$ removes *all* copies of the listed values and, in expectation, only $\approx$2 elements are dropped. Algorithm 2 collects these steps.

A spurious hit survives this $\approx$490-bit test with negligible probability, and the one true hit survives by construction (up to the s=0 case already absorbed into the data complexity in §4.1). Thus, the total cost is $2^{80} \cdot 2^{11}/160 \approx 2^{84}$ encryptions, well below the 2^{89} main loop.

From the surviving hit to the full key. After Algorithm 2, the attacker holds exactly one bridge-consistency survivor: the correct pair $(P^{(0)}, C^{(0)})$, the eight online key bytes $k_{-1}[0, 5, 10, 15]$ and $u_6[0, 7, 10, 13] = k_6[0, 7, 10, 13]$, the byte $u_5[0] = v_0 \oplus S(s)$ recovered from the bridge value s, and the stored table payload — the eleven parameter bytes of [DFJ13, §3.3] (the ten bytes $\Delta z_1[0]$, $x_2^{(0)}[0..3]$, $\Delta w_4[0]$, $z_4^{(0)}[0..3]$ of Proposition 2 plus the second active-difference byte) together with the $\approx$ 20 DDT branch bits. Algorithm 3 turns these into the full 128-bit master key K_0 at $\approx 2^{44.6}$ encryptions (cost paragraph below) — run once, more than forty bits below the online phase — and is verified end-to-end in §5.

The procedure is the key-completion step of Derbez, Fouque and Jean [DFJ13, §3.4], specialized to our setting where the parameter bytes are stored rather than re-guessed. One pass of ConstructTable over the stored payload reconstructs the 24 Demirci–Selçuk parameters $x_2^{(0)}[0..3]$, $x_3^{(0)}[0..15]$, $x_4^{(0)}[0, 5, 10, 15]$ of the reference message (and incidentally the key bytes $u_2[0, 7, 10, 13]$ and $k_3[0, 5, 10, 15]$, which we use only as a cross-check). The observation is then purely geometric: for each of the three *remaining* byte positions $i \in \{1, 2, 3\}$ of column 0 of x_5, the 24-byte function that maps $\Delta y_1[0]$ to $\Delta x_5[i]$ depends on exactly the *same* 24 parameters — all of which are now known — so the offline side of a fresh attack of the kind in [DS08] at position i is free. On the online side, peeling the δ-set ciphertexts back to $x_5[i]$ requires only five new key bytes (Table 1), and the three anti-diagonals $\{3, 6, 9, 12\}$, $\{2, 5, 8, 15\}$, $\{1, 4, 11, 14\}$ together with the already-known $\{0, 7, 10, 13\}$ cover k_6 exactly.

Table 1: Byte geometry of CompleteKey. For $x_5[i]$, the online peel reads ciphertext bytes $C[\text{idiag}(c')]$ with $c' = (-i) \bmod 4$, uses the four listed k_6 bytes and the listed u_5 byte, and compares the resulting $\Delta x_5[i]$ sequence to the one computed (for free) from the 24 parameters.

i	z_5-byte	x_6-column	new k_6 bytes	new u_5 byte
1	13	3	3, 6, 9, 12	13
2	10	2	2, 5, 8, 15	10
3	7	1	1, 4, 11, 14	7

Cost. Each of the three inner loops is 2^{40} candidates times 256 partial decryptions of five S-box lookups each, i.e. $3 \cdot 2^{40} \cdot 256 \cdot 5/160 \approx 2^{44.6}$ encryptions; this is the column-0

Algorithm 3 CompleteKey: from one bridge-consistency survivor to K_0.

Require: correct pair $(P^{(0)}, C^{(0)})$; $k_{-1}[0,5,10,15]$; $u_6[0,7,10,13]$; $u_5[0]$; the eleven stored parameter bytes and branch bits (table payload); the 256 δ-set ciphertexts $C^{(\omega)}$.
Ensure: the 128-bit master key K_0.
1: $(x_2^{(0)}[0..3],\ x_3^{(0)}[0..15],\ x_4^{(0)}[0,5,10,15]) \leftarrow \textsc{ConstructTable}(\text{params}, \text{branch})$ ▷ one pass, $\approx 2^{11}$ lookups
2: $k_6[0,7,10,13] \leftarrow u_6[0,7,10,13]$
3: **for** $i \in \{1,2,3\}$ **do**
4: compute $D_i[\eta] \leftarrow \Delta x_5[i]^{(\eta)}$ for $\eta = 1..255$ from the 24 parameters ▷ $\approx 2^{10}$ lookups, offline side known exactly
5: **for** each of the 2^{40} candidates $(k_6[\text{idiag}_{c'_i}],\ u_5[p_i])$ ▷ c'_i, p_i from Table 1 **do**
6: peel $C^{(\omega)} \to x_6[\text{col}\, c'_i] \to z_5[p_i] \to y_5[i] \to x_5[i]^{(\omega)}$ for $\omega = 0..255$
7: **if** $\left\{x_5[i]^{(\omega)} \oplus x_5[i]^{(0)}\right\}_\omega = \{D_i[\eta]\}_\eta$ as multisets **then**
8: record the five bytes; **break** ▷ expected exactly one survivor; the ordered 255-byte sequence is a >2000-bit filter
9: assemble $k_6[0..15]$ from $k_6[0,7,10,13] \cup k_6[3,6,9,12] \cup k_6[2,5,8,15] \cup k_6[1,4,11,14]$
10: $K_0 \leftarrow \textsc{KeySchedule}^{-1}(k_6)$
11: **verify** $\textsc{Enc}_{K_0}(P^{(0)}) = C^{(0)}$ and $u_5[0,7,10,13]$, $k_{-1}[0,5,10,15]$, $u_2[0,7,10,13]$, $k_3[0,5,10,15]$ all consistent with K_0; **return** K_0.

instantiation of [DFJ13, §3.4], whose generic estimate over all 16 positions (with up to 2^{32} offline re-enumeration at the 12 positions outside column 0) is $16 \cdot 2^{40} \cdot 2^8/2^5 \approx 2^{47}$ encryptions. Either way the step runs *once* and is dominated by the $2^{89.3}$ online phase by more than forty bits.

4.1.1 Overall complexity

Calculating the overall complexity of the attack depends on the exact method used to account for runtime. Under standard accounting, each unit of runtime corresponds to a single AES computation—but there is no standard convention to decide how to account for a partial round, and even less of a standard when deciding how to account for additional computation that is not AES calculations but rather operations like storing intermediate results in hashtables. The standard convention in this line of work, and the one we adopt because it is the one that makes our numbers comparable to the 2^{99} of [DFJ13], is to count *table lookups*: one AES encryption is priced at $C_{\text{AES}} = 160$ of them (ten rounds of sixteen S-boxes; the linear layer and key schedule are free), and everything the attacker does is converted into an equivalent number of 256-entry table reads. This is generous to attacks in one direction (a real machine still pays for MixColumns) and stingy in another (a real machine can evaluate sixteen S-boxes with one vector instruction); we return to the question of how far our attack sits from wall-clock reality in Appendix D.

We account for the runtime in two different ways. To begin, let us calculate what the Möbius bridge itself contributes, and to state it we must separate the number of loop iterations N from the time T. The inner loop of [DFJ13] runs $N = 2^{96}$ times (2^{72} candidate pairs, and for each a $2^8 \cdot 2^8 \cdot 2^8$ enumeration of k_{-1}, u_6, and $u_5[0]$), and each iteration costs 256 partial encryptions of five S-boxes each, 1280 lookups; their $T = 2^{99}$ is just $2^{96} \cdot 1280/160 = 2^{96} \cdot 2^3$. The bridge removes the $u_5[0]$ enumeration and so acts on N, not on the per-iteration cost: N drops from 2^{96} to 2^{88}, a clean factor of 256 regardless of any accounting measure. What it does to T then depends only on whether our per-iteration cost is comparable to the 1280 of [DFJ13] — had it been identical, the result would be exactly $2^{99}/256 = 2^{91}$. Getting there is not free (a naïve evaluation of the power-sum fingerprint is 2^{19} operations per iteration, which would surrender the whole gain and then some), but the packed power table of Appendix A.1 brings the per-iteration cost to about 1690 lookups against the 1280 of [DFJ13], and the bridge's factor of 256 then lands the

attack at $T = 2^{88} \cdot 1690/160 = 2^{91.4}$. This is the number to hold on to for a reader who distrusts the finer bookkeeping that follows.

We can do better with a finer-grained account. Our goal here will be to make each step as cheap as possible for each of the 2^{88} iterations of the loop, exploiting the enumeration order of the table (the DDT-aware Gray code of §A.2), a cache that replaces the online inverse S-boxes (§A.3), and a cheaper fingerprint (§4.2–§A.5). This brings the attack to $2^{89.3}$, a further two bits, and it is where the accounting choices actually start to matter, since these optimizations are precisely the kind of computation — amortized S-boxes, cached partial peels, wide XORs — that the convention prices most loosely.

We now walk through this second step in more detail. For this second account we resolve the per-iteration cost into two pieces. Each of the $N = 2^{88}$ iterations computes 255 values (the d_ω offline, the g_ω online) and folds them into one fingerprint, so write c_ω for the lookups spent on a single element and C_{post} for the once-per-iteration work of collapsing the 255 accumulated values into a fingerprint:

$$T \;=\; \frac{N \cdot \big(255\, c_\omega + C_{\text{post}}\big)}{C_{\text{AES}}}, \qquad N = 2^{88},\; C_{\text{AES}} = 160.$$

(In this notation the first account is $c_\omega = 6$, $C_{\text{post}} = 160$.) The offline table-build P is the same expression at the offline per-element cost, which we check comes out at or below T. The rest of this section prices the two knobs c_ω and C_{post}; Table 2 shows how each technique moves the total, with the first account as its second row.

Consider first the online per-element cost. Each ω requires peeling the ciphertext back to $v_\omega = \mathsf{MC}^{-1}(x_6)[0]$, which is four iSBOX lookups; then one L^{-1} lookup and one field inverse to form $g_\omega = (L^{-1}\Delta v_\omega)^{-1}$; then one wide XOR to fold g_ω into the packed accumulator of §A.1. Folding the inverse into the accumulator table's index, that is $4 + 1 + 1 = 6$ lookups per element — the first account's c_ω. The Gray-code walk of §A.2 then amortizes the four iSBOX lookups over the sixteen u_6 branch-bit configurations, so that on average only $(4 + 15 \cdot 1)/16 = 1.19$ of them are paid per step, and $c_\omega = 1.19 + 1 + 1 = 3.19$.

Offline the same loop is more expensive per element, because d_ω is obtained by tracing the δ-difference through the three S-box layers of rounds 2–4: about 20 active S-boxes, plus the one accumulate. But the offline enumeration also has more structure to exploit — its 20 DDT branch bits against the four online ones — and the same Gray-code walk amortizes the 20 S-boxes down to 1.06 per element, giving $c_\omega^{\text{off}} = 1.06 + 1 = 2.06$.

Finally, the combine step C_{post} is the cost of collapsing the accumulator into a fingerprint once the 255 elements have been absorbed. For the $I_{m,n}$ fingerprint this is the polynomial evaluation of Appendix A.1, about 160 lookups; for the $\chi^\star$ fingerprint it is the frame solve of §4.2, about 15.

Table 2: Per-entry cost $\ell = 255 \cdot c_\omega + C_{\text{post}}$ and resulting $T = 2^{88} \cdot \ell/160$. Techniques apply cumulatively; $N = 2^{88}$ throughout (§4.1).

technique (cumulative)	c_ω^{on}	C_{post}	ℓ	T
— (naïve pairwise $I_{m,n}$)	6	$\approx 2^{19}$	$\approx 2^{19}$	$2^{99.7}$
+ packed POW (§A.1)	6	160	1690	$2^{91.4}$
+ DDT-Gray (§A.2)	3.19	160	973	$2^{90.6}$
+ W-cache (§A.3)	1.49	160	540	$2^{89.75}$
+ $\chi^\star$ (§4.2)	1.49	15	395	$\mathbf{2^{89.30}}$
+ $\chi^{\star\prime}$ (§A.5)	1.49	8	**388**	$\mathbf{2^{89.28}}$
[DFJ13] (N=2^{96})	5	—	1280	2^{99}

A balanced variant. Above we focus on minimizing the total *time* at a fixed data budget of $D = 2^{105}$. But because $\max(D, T, M) = 2^{105}$, data remains the bottleneck, and so the

overall complexity of our attack has not improved. A standard alternative metric is to minimize the *balanced* cost $\max(D, T, M)$. We do this now.

First, observe that this style of attack has a knob that trades data for memory. Following [DFJ13, §3.3], instead of one differential trail we run K trails in parallel (different input/output byte positions for the active differences). This causes memory to grow by K (we store K tables), data to fall by K (any one trail producing a right pair suffices), while leaving the runtime unchanged (the per-pair work and the number of surviving pairs move by canceling factors of K). The balanced point of [DFJ13] uses $K \approx 2^8$, giving $(D, T, M) = (2^{97}, 2^{99}, 2^{98})$ and hence $\max(D, T, M) = 2^{99}$.

Because our bridge technique has cut T by $\approx 2^{9.7}$, there is more room to push D down before it meets T. Optimizing over the available parameters, and using a cheaper ordered fingerprint suited to the many-table setting, gives

$$\max(D, T, M) = 2^{96.3},$$

with $(D, T, M) = (2^{96.1}, 2^{96.3}, 2^{96.1})$; Appendix C derives this variant in full, and gives a second version at $\max(D, T, M) = 2^{96.9}$ that drops the fingerprint-randomness assumption entirely. This is a 2.7-bit improvement over [DFJ13] at the same success probability of $1-1/e \approx 63\%$. (The attack is probabilistic because [DFJ13] tune their data to one *expected* right pair, and at least one appears with probability $1-1/e$.)

4.2 An improved fingerprint: χ-canonicalization

The power-sum invariants $I_{m,n}$ of §4.1 are correct but are suboptimal in two ways. First, they support up to ≈14 bytes (≈110 bits) of fingerprint for the raw 255-element multiset, and slightly less for the Add-0 variant of §4.1 (§A.4); we would prefer to have a larger fingerprint. Second, even with the efficient packing of Appendix A.1 constructing these fingerprints is computationally expensive. This section develops a different fingerprint with the same invariance properties, but which is both cheaper to evaluate and supports larger fingerprints. The key idea is to *canonicalize* the data that each side holds, instead of *algebraically eliminating* the unknown byte. That is, instead of forming ratios that cancel the unknown s, we explicitly compute an affine transform that maps each fingerprint to a canonical reference frame, and then always make comparisons between these canonicalized reference frames.

Recap of what the fingerprint must satisfy. Recall that we defined d_ω as the value we can compute offline,

$$d_\omega := x_5[0]^{(\omega)} \oplus x_5[0]^{(0)} = a_\omega \oplus a,$$

and g_ω as the value that can be computed online, specifically:

$$g_\omega := \left(L^{-1}(v_0 \oplus v_\omega)\right)^{-1}.$$

Also recall the bridge identity $g_\omega = s^2 \cdot d_\omega^{-1} \oplus s$. If we write $\mathcal{D} := \{d_\omega^{-1}\}_{\omega=1}^{255}$ for the offline multiset and $G := \{g_\omega\}_{\omega=1}^{255}$ for the online one, it is easy to see that we must have the property that

$$G \;=\; \alpha \cdot \mathcal{D} \;\oplus\; \beta, \qquad \alpha := s^2,\ \beta := s,$$

where the product and addition are taken element-wise and comparison is over the multisets themselves. Because these are multisets, the lack of a shared ordering between the two sides is harmless — any symmetric function of the 255 values is computable on both (this was the key innovation of [DKS10]). However, neither side knows the value of (α, β), and so a direct multiset comparison still fails: G and $\mathcal{D}$ are equal only up to an unknown affine

map. A valid fingerprint is therefore any function Φ of a 255-element multiset over GF(256) that is invariant under the affine group

$$\mathrm{AGL}(1,256) \;=\; \big\{\, v \mapsto \alpha v \oplus \beta \;:\; \alpha \in \mathrm{GF}(256)^{\times},\ \beta \in \mathrm{GF}(256) \,\big\}.$$

That is, $\mathrm{AGL}(1,256)$ is the group of all invertible affine maps $v \mapsto \alpha v \oplus \beta$ on a GF(256) byte where α is chosen among the non-zero elements and β is any constant (and thus the set has $255 \times 256 = 65280$ possible functions). In context, the unknown s acts on the offline data via the specific element $(\alpha, \beta) = (s^2, s)$ of this group. This statement is thus slightly more general than what we require (because we have the property that $\alpha = \beta^2$), but it says Φ must be invariant under the whole group and so it doesn't matter which element s selects.

The core observation of the prior section (expressed more generally this time) is that if the fingerprint Φ were invariant over $\mathrm{AGL}(1,256)$, then we would have that $\Phi(G) = \Phi(\alpha\mathcal{D} \oplus \beta) = \Phi(\mathcal{D})$ regardless of the parameter s. The $I_{m,n}$ construction satisfied this by algebra; here we satisfy it by orbit canonicalization.

Step 1: the parity vector χ. The first step compresses the multiset to a 256-bit string. For a multiset V over GF(256) define

$$\chi(V)_d \;:=\; \big|\{\, \omega : v_\omega = d \,\}\big| \mod 2, \qquad \chi(V) \;\in\; \mathrm{GF}(2)^{256},$$

that is, bit d records whether the value d occurs an odd number of times in V. This is symmetric in the indices by construction, and it is also cheap to calculate because it costs just one XOR per element,

$$\chi \;\leftarrow\; \chi \;\oplus\; \mathbf{e}_{v_\omega} \qquad (\omega = 1, \ldots, 255),$$

where $\mathbf{e}_d$ is the unit vector with a 1 in position d.

The key question is then: how does the affine action look on χ? Let $V' = \alpha V \oplus \beta$, i.e. $v'_\omega = \alpha\, v_\omega \oplus \beta$ elementwise, and evaluate $\chi(V')$ at an arbitrary position d':

$$\begin{aligned}
\chi(V')_{d'} \;&=\; \big|\{\, \omega : v'_\omega = d' \,\}\big| \bmod 2 && \text{(def. of } \chi\text{)} \\
&=\; \big|\{\, \omega : \alpha\, v_\omega \oplus \beta = d' \,\}\big| \bmod 2 && \text{(def. of } V'\text{)} \\
&=\; \big|\{\, \omega : v_\omega = \alpha^{-1}(d' \oplus \beta) \,\}\big| \bmod 2 && (\alpha \neq 0\text{, solve for } v_\omega) \\
&=\; \chi(V)_{\,\alpha^{-1}(d' \oplus \beta)} && \text{(def. of } \chi\text{).}
\end{aligned}$$

Substituting $d' := \alpha d \oplus \beta$ gives

$$\chi(V')_{\,\alpha d \oplus \beta} \;=\; \chi(V)_{\,d} \qquad \text{for every } d \in \mathrm{GF}(256).$$

Because $d \mapsto \alpha d \oplus \beta$ is a bijection on GF(256), the above equivalence says the 256 bits of χ are just relabeled. The affine map on *values* acts on χ as a pure *permutation of bit positions*. We write $\pi_{\alpha,\beta} \colon d \mapsto \alpha d \oplus \beta$ for this index permutation and $\chi(V') = \pi_{\alpha,\beta} \cdot \chi(V)$.

For clarity, note that two different permutations are in play here, acting on different objects. Reordering the *input indices* ω — the S_{255} action that online and offline cannot synchronize — leaves χ unchanged, since a histogram does not care in which order the values were listed; this is the "symmetric in the indices" property above. But the affine map acts instead on the *output bit-positions* d of χ, and against that χ is *not* invariant. However, the discrepancy is simple: online's $\chi(G)$ and offline's $\chi(\mathcal{D})$ differ only by the unknown bit-permutation $\pi_{s^2,\,s}$. Step 1 has thus disposed of the first unknown and reduced the second to a permutation drawn from a known $255 \cdot 256$-element group; the next two steps remove it.

Step 2: solving for a canonical frame $(\alpha^\star, \beta^\star)$. A *moving frame* is a recipe that, given V, picks a specific $(\alpha^\star, \beta^\star) \in \mathrm{AGL}(1, 256)$ depending only on V, such that applying $\pi_{\alpha^\star, \beta^\star}$ to $\chi(V)$ lands on the same canonical vector regardless of which point in the orbit we started from. Concretely we need

$$\alpha^\star(\alpha V \oplus \beta) \;=\; \alpha^{-1}\,\alpha^\star(V), \quad \beta^\star(\alpha V \oplus \beta) \;=\; \beta^\star(V) \;\oplus\; \alpha^\star(\alpha V \oplus \beta)\cdot\beta,$$

because then $\pi_{\alpha^\star, \beta^\star} \circ \pi_{\alpha,\beta} = \pi_{\alpha^\star(V), \beta^\star(V)}$ collapses to the same map on both sides.

We build the frame from a handful of *single-index* power sums

$$S_k(V) \;:=\; \bigoplus_{\omega} v_\omega^k,$$

because their behavior under the group action $V \mapsto \alpha V \oplus \beta$ is known exactly. Consider first the raw fingerprint, where $|V| = 255$. By Lucas's theorem, $\binom{k}{j} \bmod 2 = 1$ iff every set bit of j is also set in k (written $j \subseteq k$), so in characteristic 2 the binomial expansion keeps only those terms:

$$\begin{aligned}
S_k(\alpha V \oplus \beta) \;&=\; \bigoplus_{\omega} (\alpha v_\omega \oplus \beta)^k && \text{(def. of } S_k\text{)}\\
&=\; \bigoplus_{\omega} \bigoplus_{j \subseteq k} \alpha^j \beta^{\,k-j}\, v_\omega^j && \text{(char-2 binomial)}\\
&=\; \bigoplus_{j \subseteq k} \alpha^j \beta^{\,k-j} \bigoplus_{\omega} v_\omega^j && \text{(swap sums)}\\
&=\; \bigoplus_{j \subseteq k} \alpha^j \beta^{\,k-j}\, S_j,
\end{aligned}$$

where $S_0 = \bigoplus_\omega 1 = |V| \bmod 2 = 1$. This transformation law is all we use below.

Choosing $\alpha^\star$: strip the scale first. We deal with the two unknowns one at a time, and start with the scale α. For that we want a quantity computed from the values that transforms by a known power of α and is *blind to* β — otherwise the shift would contaminate our estimate of the scale. Exactly such a quantity already appeared in §4.1: the pairwise-difference power sum

$$P_7(V) \;:=\; \bigoplus_{\omega < \omega'} \big(v_\omega \oplus v_{\omega'}\big)^7.$$

Its behavior under $V \mapsto \alpha V \oplus \beta$ can be read off directly: inside every difference the shift cancels,

$$(\alpha v_\omega \oplus \beta) \oplus (\alpha v_{\omega'} \oplus \beta) \;=\; \alpha\,(v_\omega \oplus v_{\omega'}),$$

so each term becomes $\alpha^7 (v_\omega \oplus v_{\omega'})^7$ and hence

$$P_7(\alpha V \oplus \beta) \;=\; \alpha^7\, P_7(V). \tag{1}$$

That is, P_7 is exactly translation-invariant and picks up a clean α^7 under scaling. Since $\gcd(7, 255) = 1$ the map $t \mapsto t^7$ is a bijection on $\mathrm{GF}(256)^\times$ (its inverse is $t \mapsto t^{73}$, because $7 \cdot 73 \equiv 1 \pmod{255}$), so provided $P_7 \neq 0$ we can set

$$\alpha^\star(V) \;:=\; P_7(V)^{-1/7} \;=\; P_7(V)^{-73}.$$

By (1), $\alpha^\star(\alpha V \oplus \beta) = (\alpha^7 P_7(V))^{-1/7} = \alpha^{-1}\alpha^\star(V)$ — the first half of the covariance equation. Concretely, if online rescales its values by its own $\alpha^\star$ it obtains

$$\alpha^\star(\alpha V \oplus \beta)\cdot(\alpha v_\omega \oplus \beta) \;=\; \alpha^{-1}\alpha^\star(V)\,(\alpha v_\omega \oplus \beta) \;=\; \alpha^\star(V)\, v_\omega \;\oplus\; \underbrace{\alpha^{-1}\alpha^\star(V)\,\beta}_{=:\,\tilde\beta},$$

which is offline's rescaled data $\alpha^\star(V)\, v_\omega$ up to a single leftover translation $\tilde{\beta}$. The unknown scale is gone; only a shift remains, and we remove it next.

Fallback when $P_7 = 0$. The same argument works for any exponent m coprime to 255: P_m is translation-invariant and scales as α^m, so $\alpha^\star := P_m^{-1/m}$ is equally valid. We use the first nonzero P_m in the fixed chain $7 \to 11 \to 13 \to 23 \to 31 \to 127$; both sides see the same zero pattern ($P_m \mapsto \alpha^m P_m$), so they always agree on which level to use. For the raw fingerprint the first three levels fail together only with probability $\approx 2^{-24}$. The Add-0 variant is why the chain continues past 13: at even $|V|$ every term of P_7, P_{11}, P_{13} carries a factor of S_1, so all three vanish together whenever $S_1 = 0$ (probability 2^{-8}), while P_{23}, P_{31}, P_{127} do not; the same S_1=0 event affects $\beta^\star$ and is handled in "Choosing $\beta^\star$" below (branch table: Appendix B).

Computing P_7 cheaply. The pairwise definition costs $\binom{255}{2}$ operations, but Reduction 1 of App. A.1 collapses it to a short polynomial in the single-index sums S_k (these are the p_k of §4.1; $S_k \equiv p_k$). Concretely, expand the pair term with the char-2 binomial: since $7 = 111_2$ every $k \in \{0, \ldots, 7\}$ is a submask, so

$$(x \oplus y)^7 \;=\; x^7 \oplus x^6 y \oplus x^5 y^2 \oplus x^4 y^3 \oplus x^3 y^4 \oplus x^2 y^5 \oplus x y^6 \oplus y^7.$$

From here we sum over unordered pairs $\{v_\omega, v_{\omega'}\}$ and group the mixed terms into complementary pairs; each such pair is a full ordered-pair sum minus its diagonal, e.g.

$$\bigoplus_{\omega<\omega'} \left(v_\omega^6 v_{\omega'} \oplus v_\omega v_{\omega'}^6\right) \;=\; \bigoplus_{\omega\neq\omega'} v_\omega^6 v_{\omega'} \;=\; \Big(\bigoplus_{\omega} v_\omega^6\Big)\Big(\bigoplus_{\omega'} v_{\omega'}\Big) \;\oplus\; \bigoplus_{\omega} v_\omega^7 \;=\; S_6 S_1 \oplus S_7,$$

and likewise $S_5 S_2 \oplus S_7$ and $S_4 S_3 \oplus S_7$ for the other two. The pure terms $x^7 \oplus y^7$ contribute nothing: each v_ω^7 occurs in $|V| - 1 = 254$ pairs, an even number, so they cancel (this is the $P_1 \equiv 0$ parity phenomenon of §4.1 again). Adding it up, three copies of S_7 collapse to one and

$$P_7 \;=\; S_1 S_6 \;\oplus\; S_2 S_5 \;\oplus\; S_3 S_4 \;\oplus\; S_7. \tag{2}$$

The even-indexed sums here cost nothing extra: in characteristic 2 squaring distributes over XOR — $(a \oplus b)^2 = a^2 \oplus b^2$, since the cross term $2ab$ vanishes — so $S_{2k} = \bigoplus_\omega (v_\omega^k)^2 = \big(\bigoplus_\omega v_\omega^k\big)^2 = S_k^2$, i.e. $S_2 = S_1^2$, $S_4 = S_1^4$, $S_6 = S_3^2$. Hence P_7 is a polynomial in the odd sums S_1, S_3, S_5, S_7 alone, and those four are the only quantities accumulated in the inner loop. (With $|V| = 256$ — the Add-0 variant — each pure term occurs an odd number of times and the parity in the derivation flips; the net effect is that the lone S_7 term drops from (2).)

Choosing $\beta^\star$: strip the leftover shift. *First*, rescale everything by $\alpha^\star$, letting $S'_k := (\alpha^\star)^k\, S_k$. By the covariance[9] of $\alpha^\star$, the primed sums are *scale-invariant*; the only thing left to remove is the leftover translation $\tilde{\beta}$ identified above. *Then*, we need a primed sum that moves under translation *by exactly* $\tilde{\beta}$, and the transformation law tells us which one: taking $k = 1$ there (its only submasks are $j \in \{0, 1\}$) gives

$$S_1 \;\longmapsto\; \alpha S_1 \oplus \beta,$$

i.e. S_1 shifts by exactly the translation applied to the data. Removing the scale as before,

$$S'_1(\alpha V \oplus \beta) \;=\; \alpha^{-1}\alpha^\star(V)\,\big(\alpha S_1(V) \oplus \beta\big) \;=\; S'_1(V) \;\oplus\; \tilde{\beta},$$

so S'_1 tracks the leftover shift precisely, and for the raw fingerprint the canonical translation is simply

$$\beta^\star \;:=\; S'_1 \;=\; \alpha^\star S_1.$$

[9] In the invariant-theory sense, not the statistical one: a *covariant* transforms by a known factor under the group action, here $\alpha^\star \mapsto \alpha^{-1}\alpha^\star$, as opposed to an *invariant*, which is fixed. The point is that $(\alpha^\star)^k$ picks up exactly the α^{-k} needed to cancel the leading α^k in S_k.

Both halves of the covariance equation now hold, and the composed map is the same on both sides: online's canonical values are $\alpha^\star(V')v' \oplus \beta^\star(V') = \left[\alpha^\star(V)v \oplus \tilde{\beta}\right] \oplus \left[S'_1(V) \oplus \tilde{\beta}\right] = \alpha^\star(V)v \oplus S'_1(V)$, exactly what offline computes from V.

The Add-0 variant needs a different $\beta^\star$. With the appended zero $|V| = 256$, so $S_0 = |V| \bmod 2 = 0$ and the k=1 line becomes $S_1 \mapsto \alpha S_1$: the translation has dropped out of S_1, and S'_1 is no longer a valid $\beta^\star$. Write $n := |V| \bmod 2$ for this bit (n=1 raw, n=0 Add-0). For $n = 0$ we canonicalize the shift with the next sum that still moves under translation, S'_3, by choosing $\beta^\star$ so that the shifted S'_3 hits a fixed target:

- *The equation.* Substituting $u := \beta^\star / S'_1$ turns the condition into the Artin–Schreier equation $u^2 \oplus u = c$ with $c := S_3/S_1^3$, an AGL-invariant quantity, so online and offline always solve the *same* equation.
- *Solvability.* In GF(2^8) this has a solution iff $\mathrm{Tr}(c) = 0$ (field trace). If $\mathrm{Tr}(c) = 1$, both sides instead solve $u^2 \oplus u = c \oplus \tau$ for a fixed τ with $\mathrm{Tr}(\tau) = 1$ (i.e. they aim S'_3 at $\tau S_1'^3$ rather than 0).
- *Two roots.* A solution comes with a partner $u \oplus 1$, giving two candidate translations $\beta^\star$ and $\beta^\star \oplus S'_1$; there is no canonical choice between them, so we keep both and take the minimum of the two resulting fingerprints (Step 3).
- *The remaining hole.* All of the above divides by S_1. When $S_1 = 0$ (probability 2^{-8}) — the same event that empties the $\alpha^\star$ chain above — $\beta^\star$ is instead canonicalized as the minimum over all 256 translations (Appendix B).

Step 3: applying the permutation in zero lookups. We now have $(\alpha^\star, \beta^\star)$ and must permute the 256 bits of χ by $d \mapsto \alpha^\star d \oplus \beta^\star$. Naïvely this is 256 field multiplications. The key observation is that *multiplication by a fixed $\alpha^\star$ in* GF(2^8) *is* GF(2)*-linear in the* 8*-bit representation*: it is an 8×8 binary matrix $M_{\alpha^\star}$ acting on the bit-vector of d. The translation $\oplus\, \beta^\star$ is an XOR of the index by a constant. Hence the index map $\pi_{\alpha^\star, \beta^\star}$ is a GF(2)-*affine* map on the 8 index bits.

A GF(2)-affine permutation of an 8-bit index is exactly a *bit-matrix-multiply/complement* (BMMC) permutation in the sense of Cormen *et al.* [CW93]. Gauss–Jordan reduces $M_{\alpha^\star}$ to a product of at most $2\binom{8}{2} + 7 = 63$ elementary row operations (XOR row i into row j; swap rows i, j). Each such operation, lifted to the 256-bit word χ, is a single shift-XOR-mask. Specifically, by "XOR index-bit i into index-bit j" we mean:

$$\chi \;\leftarrow\; \left(\chi \;\&\; \mathtt{mask}_{ij}\right) \;\oplus\; \left((\chi \;\&\; \overline{\mathtt{mask}_{ij}}) \lll 2^j\right) \;\oplus\; \ldots$$

and thus is extremely cheap to calculate on a modern processor.

The output is the canonicalized vector

$$\chi^\star \;:=\; \pi_{\alpha^\star, \beta^\star} \cdot \chi(V),$$

which is therefore identical for online and offline. When $n = 0$ the two $\beta^\star$-roots give two candidates $\chi_1^\star, \chi_2^\star = \tau_{S'_1}(\chi_1^\star)$ related by a fixed XOR-shift; the fingerprint is $\min\{\mathrm{hash}(\chi_1^\star), \mathrm{hash}(\chi_2^\star)\}$.

4.2.1 Overall complexity

As we mentioned at the beginning, this attack has two advantages: first, it supports wider fingerprints, and second, it is cheaper to calculate. We now show the latter fact is true—when we compare to the baseline $I_{m,n}$ attack of §4.1 (the packed-POW row of Table 2) before all the bells and whistles are added. If we compare the attack here to the

heavily optimized attack from before then the performance is essentially identical, but with the advantage that it does not depend on potentially unfair accounting techniques.

We mirror the accounting of §4.1.1. The setup is the same: both attacks require the same $N = 2^{88}$ iterations in the same loop; the only change is that the new fingerprint has a different per-iteration cost of $255\, c_\omega + C_{\text{post}}$. Consider first the per-element cost. The accumulator now needs, per element, the parity-vector update $\chi \leftarrow \chi \oplus \mathbf{e}_v$ together with the four odd single-index sums S_1, S_3, S_5, S_7; the even-indexed sums are free squarings and nothing else in Step 2 or 3 reads the raw elements. Packing these as one 40-byte row per field element,

$$T[v] \;=\; \big(\mathbf{e}_v \,\big\|\, v^1 \,\big\|\, v^3 \,\big\|\, v^5 \,\big\|\, v^7\big) \qquad (256 \text{ bits of } \chi \text{ plus four bytes of } S_k),$$

the entire per-element work is one indexed load and one wide XOR into a running accumulator — exactly the shape of the $I_{m,n}$ inner loop, with a 30-byte power row replaced by this 40-byte one, so the fingerprint contributes the same single lookup per element and c_ω is unchanged from Table 2: 1.49 online with the ciphertext peel of §A.3 folded in, and 1.00 offline.

Now consider the combine cost, which we will show has a substantially lower overhead. Where the $I_{m,n}$ fingerprint spent ≈160 lookups per iteration evaluating polynomials in the power sums, χ-canonicalization spends only the moving-frame solve: three logs of S_1, S_3, S_5 and one of the accumulated products to form $\log P_7$, three antilogs for the three terms of P_7, one antilog for $\alpha^\star = P_7^{-73}$, and three lookups (trace, half-trace, $\log u$) for $\beta^\star$ — eleven lookups — plus two or three more per parity for the rescaling, the two parities sharing the $(\alpha^\star, \beta^\star)$-solve because $\beta^\star$ depends only on S_3/S_1^3, in which $\alpha^\star$ cancels. The bit-permutation of Step 3 is word arithmetic and costs no lookups under the convention. In total $C_{\text{post}} \approx 15$ for both parities, against the 160 of the $I_{m,n}$ combine (and Appendix A.5 tabulates the solve down to $C_{\text{post}} = 8$). Against the baseline this is the whole story: with the same $c_\omega = 6$ per element, replacing the 160-lookup combine by a 15-lookup one takes ℓ from 1690 to 1545, i.e. T from $2^{91.4}$ to $2^{91.3}$ — cheaper, without invoking the Gray walk or the cache. Against the fully optimized $I_{m,n}$ attack the same combine saving is worth $\log_2(540/395) \approx 0.45$ bits (the step from the W-cache row to the $2^{89.30}$ and $2^{89.28}$ of Table 2), and in both accounts the total is dominated by the $255\, c_\omega$ term the two fingerprints share — hence essentially identical.

Fingerprint length. We can then take the fingerprint as the first K bytes of $\chi^\star$, choosing K to tune the false-positive rate with the memory complexity of the attack. Note that $\chi^\star$ takes $\approx 2^{240}$ *distinct* values (because there are $256 - \log_2 |\text{AGL}|$ bits of available entropy), but its *collision* entropy is only ≈ 7.79 bits per byte, because each bit of χ is the parity of one bin in a 255-into-256 occupancy and hence $\text{Bernoulli}\big(\frac{1}{2}(1 - (127/128)^{255})\big) \approx 0.432$. This per-byte figure treats the bits as independent, which they are not; nevertheless §5.1.2 measures the stored key directly and confirms the 7.79-bit slope (with a 0.7-bit deficit in the first byte for the Add-0 fingerprint). In practice we set $K = 13$ to achieve ≈ 101 bits of collision entropy and thus a per-lookup false-positive rate of $\approx 2^{89}/2^{101} = 2^{-12}$.

5 Correctness verification

It is relatively straightforward to verify that the key bridge relationships from §4 above are correct. What remains is to validate that this attack actually *works*, in the sense that it would run faster—something particularly challenging because we cannot run a $\geq 2^{90}$ experiment. Thus, the remainder of this section contains the best experiments we could come up with to check that this is the case.

5.1 Fingerprint collision entropy

5.1.1 The $I_{m,n}$ fingerprint

The false-positive analysis of §4.1 assumes that the 12-byte fingerprint $(I_{m_0,n_1}, \ldots, I_{m_0,n_{12}})$ behaves as 12 independent uniform draws from $\mathrm{GF}(256)^\times$, i.e. carries $12 \log_2 255 \approx 95.9$ bits. A hidden algebraic relation among the I_{m_0,n_j} would reduce this and inflate the false-positive count. We test for such relations by measuring the *collision entropy* of k-byte prefixes of the fingerprint over a large random sample.

Method. We draw $N = 3 \times 10^9$ independent 255-element multisets uniformly from $\mathrm{GF}(256)$, compute the fingerprint (with the degenerate-P_m fallback of §4.1), and record the first 8 bytes. After a single sort, the number C_k of pairs agreeing on the first k bytes is counted for $k = 1, \ldots, 7$, giving the estimator

$$\hat{H}_2(k) \;=\; \log_2 \frac{M(M-1)}{2C_k}, \qquad \sigma_k \;\approx\; \frac{1}{\sqrt{C_k}\,\ln 2},$$

where $M = N$ minus the 78,547 samples ($\approx 2^{-15.2}N$, as predicted) that fall into the degenerate-skip case. Under the null hypothesis of independent uniform bytes on $\mathrm{GF}(256)^\times$, $\hat{H}_2(k) = k \log_2 255$.

Result. Table 3 and Figure 2 show $\hat{H}_2(k)$ against the ideal line. Every residual lies within 1.1σ of zero; at $k = 1$ the measurement is sensitive to $\pm 10^{-8}$ bits and still consistent with exact uniformity. We conclude that no dependence among any 7 of the 12 coordinates is detectable at $N = 3 \times 10^9$, and extrapolate the full 12-byte fingerprint to the structural maximum of ≈ 95.9 bits.

Table 3: Prefix collision-entropy of the $I_{m,n}$ fingerprint over $M = 2\,999\,921\,453$ uniform-random multisets. $z = (\hat{H}_2(k) - k \log_2 255)/\sigma_k$.

k	C_k	$\hat{H}_2(k)$	$k \log_2 255$	residual	z
1	1.7646×10^{16}	7.994 353 436	7.994 353 437	-1.1×10^{-9}	−0.10
2	6.9201×10^{13}	15.988 706 858	15.988 706 874	-1.6×10^{-8}	−0.09
3	2.7138×10^{11}	23.983 058 153	23.983 060 310	-2.2×10^{-6}	−0.78
4	1.0642×10^{9}	31.977 394 984	31.977 413 747	-1.9×10^{-5}	−0.42
5	4 173 750	39.971 642 404	39.971 767 184	-1.2×10^{-4}	−0.18
6	16 233	47.977 913 135	47.966 120 621	$+1.2 \times 10^{-2}$	+1.04
7	64	55.964 555 162	55.960 474 058	$+4.1 \times 10^{-3}$	+0.02

5.1.2 The $\chi^\star$ fingerprint

The final attack keys the table on the first K=13 bytes of the canonical vector $\chi^\star$ (§4.2). Under the heuristic that each bit of χ is an independent Bernoulli$\big(\frac{1}{2}(1 - (127/128)^{255})\big) \approx$ Bernoulli(0.432) draw, one byte carries collision entropy $8 \cdot (-\log_2(p^2 + (1-p)^2)) \approx 7.79$ bits, so 13 bytes give ≈ 101 bits. The bits are certainly not independent (they are parities of the bins of a 255-into-256 occupancy, and $\chi^\star$ is χ pushed through a data-dependent bit permutation), so we test the heuristic exactly as for $I_{m,n}$: we draw $N = 3 \times 10^9$ uniform-random multisets per parity, compute the stored 13-byte key, and measure the prefix collision entropy $\hat{H}_2(k)$.

Figure 3 shows the result. For the raw 255-element multiset the measured $\hat{H}_2(k)$ lies within 0.015 bits of the $7.79k$ line for every $k \le 6$ (beyond which the collision count is too small to be informative), so, extrapolating the slope as in §5.1.1, the 13-byte key carries

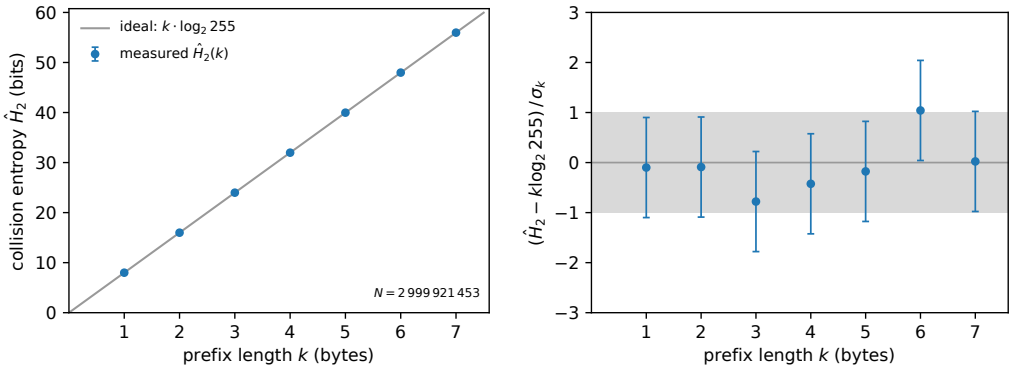

Figure 2: Left: measured $\hat{H}_2(k)$ (dots, 1σ bars) against the ideal $k \log_2 255$ (gray line). Right: standardized residuals $(\hat{H}_2(k) - k \log_2 255)/\sigma_k$; the shaded band is $\pm 1\sigma$.

≈ 101.2 bits. For the 256-element Add-0 multiset the curve is a line parallel to $7.79k$ but 0.66 bits lower, with the offset entirely in the first byte: the two Artin–Schreier roots of §4.2 give two candidate vectors, and taking their lexicographic minimum biases the leading byte. Extrapolated, the Add-0 key carries ≈ 100.6 bits. Both exceed the ≈ 97 bits needed for the 2^{-8}-per-lookup false-positive budget of §4.1.

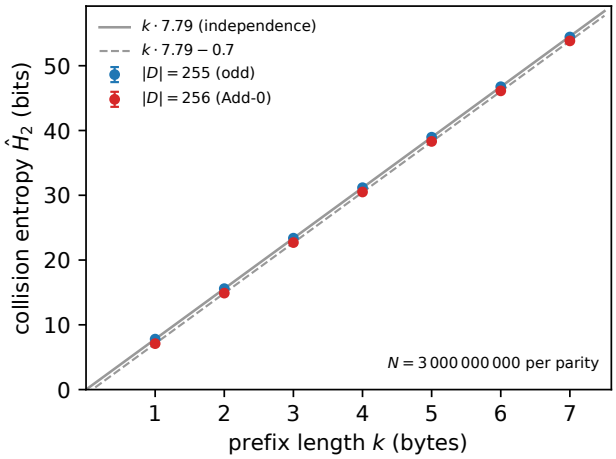

Figure 3: Prefix collision entropy $\hat{H}_2(k)$ of the stored $\chi^\star$ key (bytes 1–13) at $N = 3 \times 10^9$ uniform multisets per parity, against $7.79k$ (independence heuristic) and $7.79k - 0.7$. Error bars 1σ; only $k \leq 7$ is statistically meaningful.

5.2 Wrong-key randomization

5.2.1 The $I_{m,n}$ fingerprint

What is being tested. The attack of §4.1 has two sides. *Offline*, the precomputed table stores one 12-byte fingerprint per parameter tuple; we write I_{off} for the fingerprint of the entry that corresponds to the true key. *Online*, the attacker does not know the four bytes $k_6[0, 7, 10, 13]$, so for each candidate pair they enumerate guesses, use each guess to peel round 6, and compute a 12-byte online fingerprint I_{on} from the resulting δ-set. When the guess is correct, $I_{\text{on}} = I_{\text{off}}$ (under one of the two parities of §4.1) and the table lookup succeeds. The false-positive analysis of §4.1 assumes that when the guess is *wrong*, I_{on}

is effectively a fresh uniform draw — in particular, uncorrelated with I_{off} — so that the per-lookup hit rate is $\approx 2^{88}/2^{96} = 2^{-8}$. This section verifies that assumption directly.

Experiment. We fix one random AES-128 key, construct the δ-set at x_1, encrypt through rounds 1–6, and compute I_{off} from the true intermediate values (which we know, since we know the key). We then enumerate *all* 2^{32} values of the guess $k_6[0, 7, 10, 13]$, and for each one compute I_{on} under both parities. For every guess we record two things: whether either parity equals I_{off} (a "hit"), and the byte-Hamming distance $h = \big|\{j : I_{\text{on}}[j] \neq I_{\text{off}}[j]\}\big| \in \{0, \ldots, 12\}$.

Result. We find that exactly one of the 2^{32} guesses hits — the correct k_6. No wrong guess matches under either parity. Figure 4 histograms the Hamming distance over the remaining $2^{32}-1$ wrong guesses. If each wrong-guess fingerprint byte were an independent uniform draw from $\mathrm{GF}(256)^{\times}$, the distance would follow $\mathrm{Binom}(12, \frac{254}{255})$; the observed counts agree with this model to within 1.5σ in every populated bin (Table 4, right panel of Figure 4). Concretely: 4.097×10^9 of the 4.295×10^9 wrong guesses differ in all 12 bytes; none comes closer than $h = 7$; and among the 1020 guesses that are only *one byte* away from the correct k_6, the minimum distance is 11. A single wrong key byte therefore already randomizes the entire fingerprint. Independently, the prefix collision-entropies $\hat{H}_2(k)$ of the 2^{32} wrong-guess fingerprints are 7.9944, 15.9887, 23.9831 for $k = 1, 2, 3$ — matching the uniform-multiset measurements of §5.1.

Table 4: Byte-Hamming distance from the wrong-k_6 online fingerprint to I_{off}, over all 2^{32} guesses (one random key, seed 7). $z = (\text{obs} - \text{exp})/\sqrt{\text{exp}}$. Rows $h \leq 6$ are empty except h=0, which contains only the correct k_6.

h	observed	$\mathrm{Binom}(12, \frac{254}{255}) \cdot 2^{32}$	z
7	3	3.07	−0.04
8	518	487.24	+1.39
9	54 666	55 003.8	−1.44
10	4 190 259	4 191 287.0	−0.50
11	193 560 431	193 561 253.0	−0.06
12	4 097 048 677	4 097 046 521.0	+0.03

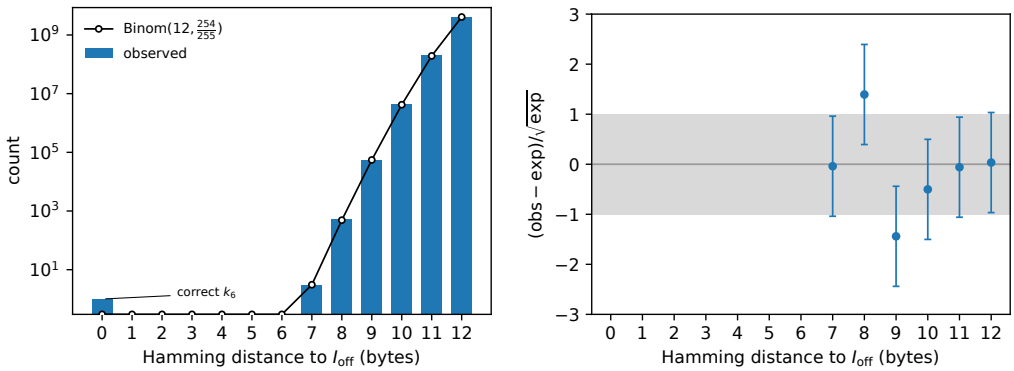

Figure 4: Left: wrong-k_6 Hamming-distance histogram (bars) against the $\mathrm{Binom}(12, \frac{254}{255})$ prediction (line), log scale; the single count at h=0 is the correct k_6, and no wrong guess comes closer than h=7. Right: standardized residuals for the populated bins; shaded band is $\pm 1\sigma$.

5.2.2 The $\chi^\star$ fingerprint

We repeat the experiment for the 13-byte $\chi^\star$ key that the final attack stores (§4.2): one fresh key, all 2^{32} guesses of $k_6[0, 7, 10, 13]$, both parities, comparing the online key to the true entry's $\chi^\star_{\text{off}}$. Again exactly one of the 2^{32} guesses hits — the correct k_6 — and no wrong guess comes closer than $h = 8$ of 13 bytes. Because the bytes of $\chi^\star$ are not uniform (§5.1.2), the null model here is not Binom$(13, \frac{254}{255})$; instead we measure the per-byte match rate q_j of the wrong-guess keys against the true entry ($q_j \in [2^{-8.9}, 2^{-7.7}]$, consistent with ≈ 7.8 bits of collision entropy per byte) and predict the Hamming histogram as the distribution of $13 - \sum_j B_j$ for independent $B_j \sim \text{Bernoulli}(q_j)$. Table 5 shows agreement to within 2σ in every bin, so the thirteen bytes of a wrong-guess $\chi^\star$ key behave as independent draws. The prefix collision entropies of the 2^{33} online keys are 7.7909, 15.5828, 23.3755 bits for $k = 1, 2, 3$, matching the uniform-multiset measurement of §5.1.2 to four decimals.

Table 5: Byte-Hamming distance from the wrong-k_6 online $\chi^\star$ key to the true entry's, over all 2^{32} guesses (one random key), against the independent-byte prediction from measured per-byte match rates. $z = (\text{obs} - \text{exp})/\sqrt{\text{exp}}$. The correct guess appears only under its matching parity and is not in this raw-parity histogram.

h	observed	predicted	z
≤ 7	0	0.0	—
8	3	4.1	−0.53
9	591	606.6	−0.63
10	64 163	64 670.9	−2.00
11	4 662 661	4 663 299.6	−0.30
12	203 933 208	203 930 339.1	+0.20
13	4 086 306 670	4 086 308 375.6	−0.03

5.3 End-to-end key recovery on SR(7,2,2,6)

Sections 5.1–5.2 validate the bridge identity and fingerprint in isolation. Here we validate the *entire* attack pipeline — structure, ciphertext filter, DDT-constrained outer-key guesses, δ-set construction, $\chi^\star$ lookup into a fully enumerated table of [DFJ13], and key-schedule solve — by running it to completion on a scaled-down cipher.

The cipher. We use the small-scale AES variant SR$(7, 2, 2, 6)$ of [CMR05]: seven rounds over a 2×2 state of 6-bit cells in GF(2^6) = GF$(2)[x]/(x^6+x+1)$. The S-box is $S(t) = L(t^{-1}) \oplus \texttt{2A}_{16}$ with L the 6-bit circulant $(1, 1, 0, 1, 0, 0)$; ShiftRows swaps the two row-1 cells; MixColumns is the self-inverse MDS matrix circ$(2, 3)$. Block and key are 24 bits. This is the smallest AES-shaped cipher on which the parameter space of [DFJ13] is enumerable *and* the $\chi^\star$ fingerprint is wide enough to filter against it: the 2×2 analogue of their 10-byte parameter set is the 6-cell tuple ($\Delta y_1[0]$, $x_2[0{,}1]$, $\Delta w_4[0]$, $z_4[0{,}1]$) plus ≤ 4 branch bits at x_3, giving $\approx 2^{36.05}$ feasible d-sequences, while $\chi^\star$ carries $\approx 64 - \log_2 |\text{AGL}(1, 64)| \approx 52$ bits. (The $I_{m,n}$ fingerprint over GF(2^6) has only 4 usable ratios = 24 bits, which is why we use $\chi^\star$ here.)

Offline. We enumerate all $63^2 \cdot 64^4 \approx 2^{36}$ parameter tuples and, for each feasible branch choice, run the 2×2 ConstructTable (verified against cipher internals on random right pairs, with zero mismatches) to obtain the 64-element sequence $d_\omega = \Delta x_5[0]_\omega$. The $\chi^\star$ fingerprint of $\{d_\omega^{-1}\}$ is computed for both parities (§4.1) and inserted into a 2^{40}-bit Bloom filter (k=5). The build touches 7.13×10^{10} d-sequences in 77 min on 32 cores

(1.54×10^7 seq/s); the resulting filter has load 0.471 and single-parity false-positive rate $p_{\mathrm{FP}} = 0.471^5 \approx 2.3\%$.

Online. For each of eight random 24-bit keys we encrypt at most three diagonal structures of 2^{12} plaintexts and run the §4.1 inner loop exactly as written, with e=6 in place of e=8: collect candidate pairs with $\Delta C[1]{=}\Delta C[2]{=}0$; per pair, DDT-constrain $k_{-1}[0,3]$ (via $\Delta x_1[1]{=}0$) and $k_6[0,3]$ (via $\Delta z_5[1]{=}0$); per surviving guess, build the 64-element δ-set, compute g_ω and $\chi^\star$ under both parities, and query the Bloom filter. *No $u_5[0]$ loop appears anywhere.* A Bloom hit fixes $(k_{-1}[0], k_{-1}[3], k_6[0], k_6[3])$; the remaining 12 key bits are recovered by key-schedule inversion and confirmed with two trial encryptions.

Result. All eight keys are recovered (Table 6). Across the 14 structures processed, the mean per-structure lookup count is $2^{22.96}$ and the Bloom-hit rate is 0.0460, matching the two-parity prediction $1-(1-p_{\mathrm{FP}})^2 = 0.0459$ (Figure 5a). The baseline of [DFJ13], which adds an inner $u_5[0]$ loop, performs $2^e{=}64\times$ as many lookups per structure for the same outcome (Figure 5b). Structures without a right pair (6 of 14) produce only false-positive hits and no key, exactly as expected.

Table 6: End-to-end key recovery on SR$(7,2,2,6)$ against the fully enumerated 2^{36}-entry table of [DFJ13] (128 GB Bloom, p_{FP}=2.3%). "Struct." is the number of 2^{12} structures tried before the key is found; χ-lookups and hits are totals across those structures.

seed	key	struct.	cand. pairs	χ-lookups	Bloom hits	time (s)
1	`27 06 29 33`	1	2 081	8 306 544	382 391	48.0
2	`3a 3f 04 0f`	2	4 104	16 406 528	753 305	91.8
3	`3a 11 18 28`	1	2 085	8 256 016	380 065	48.3
4	`1d 33 02 1e`	2	4 095	16 275 120	749 909	91.9
5	`1b 1d 2a 08`	2	4 082	16 422 928	755 799	92.2
6	`3d 39 18 31`	1	1 945	7 736 976	356 304	44.6
7	`35 03 3b 2f`	2	4 191	16 546 800	762 670	92.4
8	`38 18 22 3d`	3	6 083	24 344 288	1 120 620	134.2
per-structure mean			2 048	$2^{22.96}$	$2^{18.53}$	45.9

5.4 End-to-end attack verification for full AES

It would be computationally intractable to run the full attack end-to-end, as this would require $\approx 2^{105}$ queries and $\approx 2^{89}$ time and memory. We therefore run the attack end-to-end on the real cipher — full AES-128, real key schedule, seven rounds — but permit the attacker to "cheat" in two places. Both cheats reduce the computational complexity in a way that minimizes the risk that this cheating hides a potential flaw.

- **Don't instantiate the offline table ahead of time.** The full table exists only so that the attacker, who does not know which of the 2^{88} parameter tuples is realized by the key, can look up all of them. In a known-key harness we instead construct the *one* true entry — from the reference message's round-2/3/4 state bytes, via the same keyless $e \mapsto \Delta x_5$ map that the real enumeration evaluates, with no key byte used. This is trivially safe for *completeness*. We lose *soundness*, and instead rely on the computational measurements of §5.1.
- **Use key knowledge to find the right pair.** Honest data collection needs 2^{105} chosen plaintexts to see one right pair; the harness instead designates a reference message using the planted key. This is safe because everything *upstream* of the designated pair (i.e., structure generation and the 96-bit ciphertext filter) is inherited

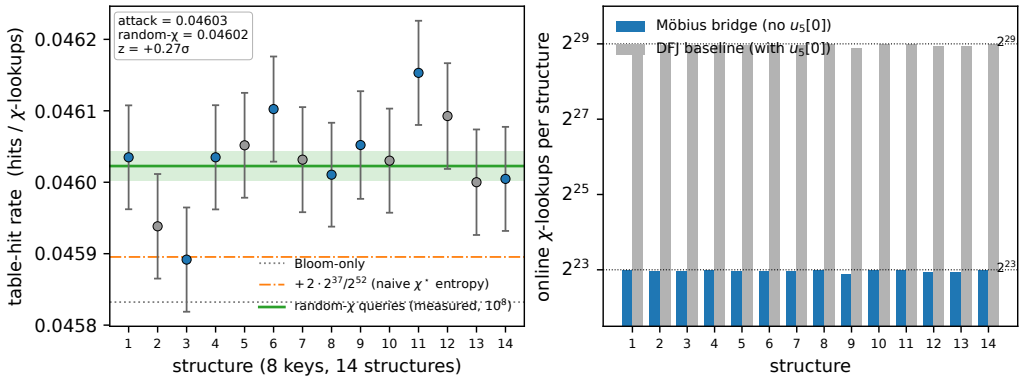

Figure 5: End-to-end SR$(7, 2, 2, 6)$ run, 14 structures over 8 keys. **(a)** Bloom-hit rate per structure; filled markers contain a right pair (and recover the key), gray markers do not. Both sit on the two-parity false-positive line, confirming that wrong (pair, k_{-1}, k_6) guesses produce $\chi^\star$ values uncorrelated with the table. **(b)** Online χ-lookups per structure: the Möbius bridge (no $u_5[0]$) sits at 2^{23}, the baseline of [DFJ13] at 2^{29} — the expected 2^e=64× gap, which is the e=6 analogue of the 2^8 factor in §4.1.

verbatim from [DFJ13] and adds no new correctness risk. We do not consult the key once we have built this structure.

Experiment. Each trial plants a fresh random 128-bit key and proceeds with the attacker's view only: build the δ-set of 256 chosen plaintexts; for every candidate $\big(k_{-1}[0, 5, 10, 15], u_6[0, 7, 10, 13]\big)$ guess, peel the ciphertexts, form $\{g_\omega\}$ and both parity fingerprints, and probe the table; dispatch every hit through Algorithm 2; from the survivor, recover $u_5[0] = v_0 \oplus S(s)$, run the three Demirci–Selçuk completion tests for the remaining twelve bytes of k_6, invert the key schedule, and confirm by trial encryption.

Result. Over 50 fresh AES-128 keys (offline decoy width $W = 2^{24}$; a grid of 128×128 wrong online guesses plus the true one, in both parities, i.e. 33 282 table lookups per key) the harness recovers the planted master key in 50/50 trials, confirmed by trial encryption. The 1 664 100 lookups produce zero fingerprint false-positive hits and zero spurious survivors of Algorithm 2; in every trial the true hit is dispatched, the recovered $s = x_5[0]^{(0)}$ and $u_5[0]$ are correct, and each of the three Demirci–Selçuk completion rows returns exactly one candidate. No key had $s = 0$ (the predicted 2^{-8} failure mode of §4.1), so no trial failed.

5.5 A machine-verified false-positive bound

The accounting in §4.1 bounds false positives under the heuristic that a wrong-key fingerprint is uniformly random. *Under this uniform randomness assumption*, we have formalized the property and proven it correct in Lean [MU21].

Informal statement. Let F be any 256-element field (AES's GF(2^8) is one). Draw $g = (g_1, \ldots, g_{255})$ uniformly from F^{255}. Let $\chi(g) \in (\mathbb{Z}/2)^F$ be the parity vector $(\chi(g)_d = 1$ iff the value d occurs an odd number of times among the $g_\omega)$, and let $\chi^\star(g)$ be its orbit under the affine group $\mathrm{AGL}(1, F) = \{v \mapsto \alpha v + \beta : \alpha \neq 0\}$ acting on bit positions. Then for any set T of orbits,

$$\Pr\big[\chi^\star(g) \in T\big] \;\leq\; |T| \cdot 2^{-200}.$$

With the paper's parameters ($|T| = 2^{89}$ stored fingerprints, 2^{88} online lookups) the expected number of *full-orbit* collisions is $\leq 2^{88+89-200} = 2^{-23}$. This is *not* the attack's false-positive count — see the scope discussion below.

Scope. We cannot directly prove the property that we actually need. Two caveats separate this theorem from that property:

First, *truncation*: the theorem bounds collisions of the *full* 256-bit orbit $\chi^\star$, whereas the table of §4.2 is keyed on a K=13-byte prefix. A prefix carries at most the measured ≈7.79 bits/byte of collision entropy, so the operative per-lookup false-positive rate is the $\approx 2^{-12}$ used throughout the accounting — the 2^{-200} does not, and cannot, apply to the truncated key.

Second, *uniformity*: the theorem is a statement about the uniform distribution on F^{255}. Its application to the attack relies on the standard wrong-key-randomization heuristic — that under an incorrect key guess the 255 values g_ω are distributed as independent uniform bytes. The heuristic is the same one used by [DKS10, DFJ13]; it is not proven here, and our evidence for it is the empirical 2^{32}-key test of §5.2.

The Lean proof therefore rules out a structural flaw in the orbit map itself (e.g. a hidden collision family in $\chi^\star$); the false-positive accounting of the attack rests on the measurements, not on the formal bound.

The Lean statement. We prove the following formal statement:

```
variable (F : Type) [Field F] [Fintype F] [DecidableEq F]

abbrev Tuple     := Fin 255 → F            -- 255 field elements
abbrev ParityVec := F → ZMod 2             -- one bit per field element

def chi (g : Tuple F) : ParityVec F :=     -- χ(g)_d = (#{ω | g_ω = d}) mod 2
  fun d => ((univ.filter (fun ω => g ω = d)).card : ZMod 2)

structure AGL where                        -- v ↦ α·v + β,  α ≠ 0
  α : F;  β : F;  hα : α ≠ 0

def aglAct (f : AGL F) (v : ParityVec F) : ParityVec F :=
  fun d => v (f.α⁻¹ * (d - f.β))           -- permute bit-positions by f⁻¹

def AGLOrbit (v w : ParityVec F) : Prop := ∃ f, w = aglAct F f v

def chiStar (g : Tuple F) : Quot (AGLOrbit F) :=
  Quot.mk _ (chi F g)                      -- χ*(g) = AGL-orbit of χ(g)

theorem chi_fp (hF : Fintype.card F = 256)
    (T : Finset (Quot (AGLOrbit F))) :
    Nat.card {g : Tuple F // chiStar F g ∈ T} * 2 ^ 200
      ≤  T.card * 256 ^ 255
```

The conclusion is the integer inequality $\#\{g : \chi^\star(g) \in T\} \cdot 2^{200} \leq |T| \cdot 256^{255}$; dividing both sides by $256^{255} = |F^{255}|$ gives the probability statement above.

Proof outline. (i) A character-sum identity over $(\mathbb{Z}/2)^F$ gives $2^{256} \cdot \#\{g : \chi(g) = v\} = \sum_{S \subseteq F} (\pm 1)\,(256 - 2|S|)^{255}$ exactly. (ii) The triangle inequality and grouping by $|S|$ yield $\#\{g : \chi(g) = v\} \leq 2^{-256} B_\chi$ with $B_\chi := \sum_k \binom{256}{k} |256 - 2k|^{255}$. (iii) AGL is a group and acts on parity vectors; each $\chi^\star$-fiber is one orbit of size $\leq |\mathrm{AGL}| = 256 \cdot 255$. (iv) The closed inequality $256 \cdot 255 \cdot B_\chi \cdot 2^{200} \leq 256^{255} \cdot 2^{256}$ is verified by direct evaluation. (v) A union bound over T.

5.6 Numerically estimating the computational complexity

Every complexity in this paper is counted in table lookups, with one AES encryption priced at 160 of them. That is a convention, and it is worth asking how far it sits from what a processor actually does. We therefore implemented both our attack and that of [DFJ13] and timed them on the same machine (Appendix D): the full pipeline on a small cipher, and the two units that the AES-128 count multiplies out — one offline table entry and one online candidate.

The convention is mildly optimistic for us. Our online candidate is 395 counted lookups (Table 2) but 649 measured cycles, so a lookup in our inner loop is worth about 1.6 cycles rather than the one the convention implicitly assumes; read in cycles, our T would be about 0.7 bits higher.

Carrying the measured constants through to the whole attack, the speedup we actually observe is about a bit smaller than the counted ledger predicts — $2^{6.84}$ against [DFJ13] with our own optimizations applied to their attack, and $2^{8.50}$ against their attack as published, where Table 2 would give 2^{8} and $2^{9.72}$. Appendix D attributes the difference to our measured online kernel not yet including the W-cache of §A.3, and to our fingerprint executing more counted lookups per cycle than the loop it replaces. The improvement survives in every unit we tried: priced in software T-table encryptions our total is $2^{90.4}$, against the counted $2^{89.28}$.

6 Discussion

The technical contributions in this paper were developed by a language model, including the Möbius bridge technique, the optimization techniques, and the correctness arguments. The human authors on this paper have validated the results and believe them to be correct by designing the numerical calculations and asking a language model to perform the experiments. We make all code to reproduce the experiments in this paper available at https://github.com/anthropics/cryptography-research-demo.

We believe that further exploring the direction of LLM-assisted cryptography research is interesting—particularly in cases where the attacks developed are computationally intractable to implement and require that the language model demonstrate its correctness through a combination of approaches. One direction we believe to be particularly important is to formalize the attack techniques used in the cryptographic community. If it were possible to automatically check the correctness of an attack without having to carry out the algorithm, we could have produced this paper an order of magnitude faster: compared to the tens of hours it took a language model to produce the core idea of this paper, it took the authors of this paper hundreds of hours to validate the results to a sufficient degree that we were confident in their correctness.

Expanding beyond cryptography, we see further applications to LLM-assisted research in security, as long as it is done carefully and with appropriate rigor. We see a future where research on verifiable topics proceeds significantly faster than research on other domains, and so an important area of work will be developing ways to formally verify previously-unverifiable problems.

Acknowledgements

We are grateful to Orr Dunkelman, Patrick Derbez, Jérémy Jean, Eyal Ronen, Nathan Keller, and Adi Shamir for comments on an early draft of this paper. The technical results in this paper were performed by Claude Mythos Preview; we have published the chain of thought where Claude discovered this idea at https://anthropic.com/document/aes_mobius_bridge_cot.pdf.

References

[BKR11] Andrey Bogdanov, Dmitry Khovratovich, and Christian Rechberger. Biclique cryptanalysis of the full aes. In *International conference on the theory and application of cryptology and information security*, pages 344–371. Springer, 2011.

[BODK+20] Achiya Bar-On, Orr Dunkelman, Nathan Keller, Eyal Ronen, and Adi Shamir. Improved key recovery attacks on reduced-round aes with practical data and memory complexities: A. bar-on et al. *Journal of Cryptology*, 33(3):1003–1043, 2020.

[BS91] Eli Biham and Adi Shamir. Differential cryptanalysis of des-like cryptosystems. *Journal of CRYPTOLOGY*, 4(1):3–72, 1991.

[CMR05] Carlos Cid, Sean Murphy, and Matthew JB Robshaw. Small scale variants of the aes. In *International Workshop on Fast Software Encryption*, pages 145–162. Springer, 2005.

[CW93] Thomas H Cormen and Leonard F Wisniewski. Asymptotically tight bounds for performing bmmc permutations on parallel disk systems. In *Proceedings of the fifth annual ACM symposium on Parallel Algorithms and Architectures*, pages 130–139, 1993.

[DBN+01] Morris J Dworkin, Elaine Barker, James R Nechvatal, James Foti, Lawrence E Bassham, E Roback, James F Dray Jr, et al. Advanced encryption standard (aes). 2001.

[DF13] Patrick Derbez and Pierre-Alain Fouque. Exhausting demirci-selçuk meet-in-the-middle attacks against reduced-round aes. In *International workshop on fast software encryption*, pages 541–560. Springer, 2013.

[DFJ13] Patrick Derbez, Pierre-Alain Fouque, and Jérémy Jean. Improved key recovery attacks on reduced-round aes in the single-key setting. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 371–387. Springer, 2013.

[DKS10] Orr Dunkelman, Nathan Keller, and Adi Shamir. Improved single-key attacks on 8-round aes-192 and aes-256. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 158–176. Springer, 2010.

[DR98] Joan Daemen and Vincent Rijmen. The block cipher rijndael. In *International Conference on Smart Card Research and Advanced Applications*, pages 277–284. Springer, 1998.

[DR02] Joan Daemen and Vincent Rijmen. *The design of Rijndael*, volume 2. Springer, 2002.

[DS08] Hüseyin Demirci and Ali Aydın Selçuk. A meet-in-the-middle attack on 8-round aes. In *International Workshop on Fast Software Encryption*, pages 116–126. Springer, 2008.

[LJ16] Rongjia Li and Chenhui Jin. Meet-in-the-middle attacks on 10-round aes-256. *Designs, Codes and cryptography*, 80(3):459–471, 2016.

[LJW14] Leibo Li, Keting Jia, and Xiaoyun Wang. Improved single-key attacks on 9-round aes-192/256. In *International Workshop on Fast Software Encryption*, pages 127–146. Springer, 2014.

[MU21] Leonardo de Moura and Sebastian Ullrich. The lean 4 theorem prover and programming language. In *International Conference on Automated Deduction*, pages 625–635. Springer, 2021.

A Performance of fingerprints

A.1 Evaluating the fingerprint in one lookup per element

Section 4.1 defined the fingerprint $I_{m,n} = P_m^n / P_n^m$, but did not address the critical question of how to efficiently calculate this sum. A direct computation of $P_m = \bigoplus_{\omega<\omega'} (g_\omega \oplus g_{\omega'})^m$ sums over $\binom{255}{2} \approx 2^{15}$ pairs for each of the 13 exponents — about 2^{19} field operations per lookup, which would dominate the attack. This subsection reduces that cost to one table lookup per element plus a constant per-entry combine step, in three reductions.

Reduction 1: P_m as a polynomial in single-index sums. Define the *single-index* power sums

$$p_r := \bigoplus_{\omega=1}^{255} g_\omega^r, \qquad r \geq 1 \qquad \big(\text{offline: } q_r := \textstyle\bigoplus_\omega d_\omega^{-r}\big).$$

We will show that each pairwise sum P_m is a short polynomial in these p_r. The derivation has four steps; we carry $m = 7$ as the running example.

Step (i): the char-2 binomial. Over GF(2^8) the binomial coefficients are taken mod 2, and by Lucas' theorem $\binom{m}{k} \equiv 1 \pmod 2$ exactly when every 1-bit of k is also a 1-bit of m (write $k \subseteq m$). Hence

$$(x \oplus y)^m = \bigoplus_{k \subseteq m} x^k \, y^{m-k}.$$

For $m = 7 = \mathtt{111}_2$ every $k \in \{0, \ldots, 7\}$ is a submask, so all eight terms survive.

Step (ii): substitute and swap sums.

$$\begin{aligned} P_m &= \bigoplus_{\omega<\omega'} (g_\omega \oplus g_{\omega'})^m && \text{(definition)} \\ &= \bigoplus_{\omega<\omega'} \bigoplus_{k\subseteq m} g_\omega^k \, g_{\omega'}^{m-k} && \text{(step (i))} \\ &= \bigoplus_{k\subseteq m} \underbrace{\bigoplus_{\omega<\omega'} g_\omega^k \, g_{\omega'}^{m-k}}_{=:\, T_k} && \text{(swap order of XOR).} \end{aligned}$$

Step (iii): pair T_k with T_{m-k} and factor. Since m is odd, $k \neq m-k$ for every submask, and $k \subseteq m$ iff $m-k \subseteq m$ (because $m-k = m \oplus k$ when $k \subseteq m$). Adding the two partners turns the unordered-pair sum into an ordered-pair sum, which then factors:

$$\begin{aligned} T_k \oplus T_{m-k} &= \bigoplus_{\omega<\omega'} \big(g_\omega^k g_{\omega'}^{m-k} \oplus g_\omega^{m-k} g_{\omega'}^k\big) && \text{(group the partners)} \\ &= \bigoplus_{\omega\neq\omega'} g_\omega^k \, g_{\omega'}^{m-k} && \text{(unordered} \to \text{ordered)} \\ &= \Big(\bigoplus_\omega g_\omega^k\Big)\Big(\bigoplus_{\omega'} g_{\omega'}^{m-k}\Big) \oplus \bigoplus_\omega g_\omega^m && \text{(all ordered = product} \oplus \text{diagonal)} \\ &= p_k \, p_{m-k} \oplus p_m && \text{(def. of } p_r\text{).} \end{aligned}$$

Step (iv): collect the pairs. The submasks of m split into $2^{\mathrm{wt}(m)-1}$ disjoint pairs $\{k, m-k\}$. The pair $\{0, m\}$ contributes $p_0 \, p_m \oplus p_m$; with $|V| = 255$ odd we have $p_0 =$

$\bigoplus_\omega 1 = 1$, so this term is $p_m \oplus p_m = 0$. Summing the remaining $2^{\mathrm{wt}(m)-1} - 1$ pairs:

$$P_m \;=\; \bigoplus_{\substack{k \subseteq m \\ 0<k<m-k}} p_k\, p_{m-k} \;\oplus\; p_m \qquad (m \text{ odd},\ |V| \text{ odd}), \tag{3}$$

where the lone p_m is the XOR of an odd number $(2^{\mathrm{wt}(m)-1} - 1)$ of copies of p_m from step (iii). For $m = 7$ the three interior pairs give

$$P_7 \;=\; p_1 p_6 \;\oplus\; p_2 p_5 \;\oplus\; p_3 p_4 \;\oplus\; p_7.$$

The identical derivation with q_r in place of p_r yields Q_m.

Reduction 2: even-indexed p_r are free (Frobenius). Squaring is the Frobenius endomorphism on $\mathrm{GF}(2^8)$, so

$$p_{2r} \;=\; \bigoplus_\omega g_\omega^{2r} \;=\; \Big(\bigoplus_\omega g_\omega^{r}\Big)^2 \;=\; p_r^2,$$

i.e. every even-indexed sum is a single squaring of an odd-indexed one. Only *odd* r need be accumulated. Rewriting the example, $P_7 = p_1 p_3^2 \oplus p_1^2 p_5 \oplus p_3 p_1^4 \oplus p_7$ uses only p_1, p_3, p_5, p_7. Across all 13 exponents the set of odd r that actually appear — one representative per Frobenius coset of submasks — is small; write $\mathcal{R}$ for this set; for the 13-exponent list of §4.1, $|\mathcal{R}| = 30$.

Reduction 3: a packed power table. Each p_r is a 255-term XOR; computing $|\mathcal{R}|$ of them separately would still be $|\mathcal{R}|$ lookups per element. Instead, precompute the $256 \times |\mathcal{R}|$-byte table

$$\mathrm{POW}[g] \;:=\; \big(g^r\big)_{r\in\mathcal{R}} \qquad (\text{one } |\mathcal{R}|\text{-byte row per field element}),$$

and maintain a running $|\mathcal{R}|$-byte accumulator $\mathbf{p} = (p_r)_{r\in\mathcal{R}}$. The inner loop is then

$$\text{for } \omega = 1, \ldots, 255: \qquad \mathbf{p} \;\texttt{\^{}=}\; \mathrm{POW}[\,g_\omega\,],$$

a single indexed load and a single wide XOR per element. With $|\mathcal{R}| = 30$ the row fits one 256-bit AVX2 register. Offline uses the analogous table $\mathrm{POWINV}[d] := (d^{-r})_{r\in\mathcal{R}}$, whence the name.

Per-entry combine. After the loop, the accumulator holds every odd-indexed p_r. The remaining work is independent of $|V|$: square up to the needed even indices, evaluate (3) for each of the 13 exponents ($\sum_m \big(2^{\mathrm{wt}(m)-1}-1\big) \approx 100$ field multiplications), and form the 12 ratios $I_{m_0,n} = P_{m_0}^n / P_n^{m_0}$. Carrying the multiplications in log/antilog form, the combine step is ≈ 160 table lookups per entry. For the generic selection — the first 13 admissible exponents of $\mathcal{E}$ (§4.1), i.e. $\{7, 11, 13, 19, 23, 29, 31, 37, 43, 47, 53, 59, 61\}$ — the product count is exactly 103.

Cost.

	per ω	per entry
$g_\omega = (L^{-1}(v_0 \oplus v_\omega))^{-1}$	2	—
$\mathbf{p}$ ^= $\mathrm{POW}[g_\omega]$	1	—
combine $\mathbf{p} \to \{P_m\} \to \{I_{m_0,n}\}$	—	≈ 160
total	3	+160

i.e. $\approx 255 \cdot 3 + 160 \approx 925$ table lookups per fingerprint — roughly 8 seven-round encryptions — versus $\approx 2^{19}$ for the naïve pairwise sum. This count is the *fingerprint* cost only; Table 2's

c_ω additionally includes producing v_ω from the ciphertext (four iSBOX), and folds the explicit inverse into POWINV, giving $c_\omega = 4{+}1{+}1 = 6$. The Add-0 second fingerprint of §4.1 adds only the combine step: appending $g = 0$ leaves every p_r unchanged ($0^r = 0$), so $P'_m = P_m \oplus p_m$ and the second set of ratios costs a further ≈ 60 lookups.

A.2 Amortizing the S-box cost: a DDT-aware Gray code

After §A.1 both phases share the loop structure, where for each of 2^{88} entries, and for $\omega = 1$ to 255 we compute

$$d_\omega \text{ (or } g_\omega), \quad \mathbf{p} \mathbin{\hat{}}= \text{POWINV}[\cdot] \;\to\; I_{m,n},$$

but the per-ω S-box cost differs sharply between them. *Offline*, d_ω is obtained by tracing the δ-difference through three S-box layers (rounds 2–4): $4 + 16 + 4 = 24$ active bytes, of which the first four are fixed by the outer base, leaving ≈ 20 S-boxes/ω. *Online*, g_ω requires only peeling the last round at four anti-diagonal bytes plus one L^{-1} lookup: $\approx 5/\omega$ — the same per-element cost as [DFJ13]. This subsection and the next attack the S-box term by walking the 2^{88} entries in an order where consecutive entries differ in a single S-box input; the technique is described for the offline side (where the 20 branch bits give the larger reduction, $20 \to 1.06$) and then applied online ($5 \to 1.5$).

The DDT pairing property. For any nonzero $\Delta, \Gamma \in \text{GF}(2^8)$, the equation

$$S(x) \;\oplus\; S(x \oplus \Delta) \;=\; \Gamma \tag{4}$$

has zero, two, or (for one Γ per Δ) four solutions in x. In every nonzero case the roots partition into Δ-pairs, all sharing the same output difference, and they are $\{x,\, x \oplus \Delta\}$ — each other's XOR-partner. (Because $S(t) = L(t^{-1}) \oplus c$ with $0^{-1} := 0$, for $x \notin \{0, \Delta\}$ (4) reduces to $x(x \oplus \Delta) = \text{const}$, a quadratic over $\text{GF}(2^8)$ with at most two roots, symmetric in $x \leftrightarrow x \oplus \Delta$; the unique $\Gamma = S(0) \oplus S(\Delta)$ additionally admits $\{0, \Delta\}$, giving four.) We call each such pair *DDT-partners*; by construction they yield the *same* output difference Γ.

Branch bits. Recall [DFJ13, Prop. 2] that the offline enumeration fixes the 11 parameter bytes, then derives the full middle state by solving (4) at each of the 16 bytes of round 3 (input difference $\Delta x_3[j]$ from the forward parameters, output difference $\Delta y_3[j]$ from the backward ones), and again at the 4 active bytes of round 4. Each solve contributes a (almost) *binary* choice between the two DDT-partners. Call these *branch bits*: 16 at the x_3 layer, 4 at the x_4 layer. The 2^{88} entries thus reparameterize as

$$2^{88} \text{ entries} \;=\; 2^{68} \text{ base choices} \;\times\; 2^{20} \text{ branch-bit strings},$$

where the base fixes everything except which partner is taken at each of the 20 spots.

More precisely: the AES DDT takes values in $\{0, 2, 4\}$. A 0 means the current base is inconsistent at that position and is pruned. A 4 — which occurs with conditional probability $1/127$ per position, so at $\approx 14.6\%$ of valid bases — can be handled either way:

- *Mixed-radix.* Assign the four-solution position two branch bits; all four roots share the same output difference Γ, so flipping either bit is exactly as cheap as in the two-root case. The walk becomes a mixed-radix Gray code and the leaf count per base fluctuates around 2^{20}, slightly improving the per-entry amortization of §A.3.
- *Skip.* Discard the $\approx 14.6\%$ of bases with any four-solution position. The corresponding e-sequences are never tabulated, so a right pair whose differential lands in such a base will not match; this is compensated by a factor $1/0.854 \approx 1.17$ in the data ($+0.23$ bits on D), with no change to T.

We adopt the mixed-radix option and henceforth speak of "≈ 20 branch bits"; the $2^{68} \times 2^{20}$ split above is an average.

Why flipping one branch bit is cheap. Flip the bit at $x_3[j]$: the value swaps $x_3[j] \leftrightarrow x_3[j] \oplus \Delta x_3[j]$. Because the two are DDT-partners, the output difference $\Delta y_3[j]$ is *unchanged* — and every quantity downstream that was computed from differences $(\Delta z_3, \Delta w_3, \Delta x_4, \dots)$ is therefore unchanged too. Only the single absolute value $x_3[j]$ moved. That value reaches each $d_\omega = \Delta x_5[0]$ through exactly two further S-boxes (round 3 then round 4), so updating all 255 d_ω after the flip costs 2 S-boxes per ω. For a flip at the x_4 layer, only one S-box (round 4) lies between the changed byte and d_ω: 1 S-box per ω.

The Gray-code walk. A binary Gray code lists all 2^n n-bit strings so that consecutive strings differ in exactly one bit. Walking the 2^{20} branch-bit strings in Gray order, with the four cheap (x_4-layer) bits innermost, gives

$$\begin{aligned} \text{amortized S-boxes per } \omega &= \frac{15 \cdot 1 + 1 \cdot 2}{16} \qquad \text{(15 of every 16 steps flip a cheap bit)} \\ &= 1.0625. \end{aligned}$$

The cold start (20 S-boxes/ω, once per 2^{20} steps) is negligible.

Online. The online phase has only the four u_6 DDT branch bits to walk (flipping a k_{-1} bit changes the entire δ-set, so those four bits do not amortize the S-box cost). Gray-walking the four u_6 bits gives $(4 + 15 \cdot 1)/16 = 1.19$ iSBOX/ω, plus the L^{-1} and POWINV lookups, i.e. $c_\omega \approx 3.19$.

Cost. Offline $20 \to 1.06$ gives $P \approx 2^{88}\big(255 \cdot (1.06{+}1) + 160\big)/160 \approx 2^{90.1}$; online $5 \to 3.19$ gives $T \approx 2^{88}\big(255 \cdot 3.19 + 160\big)/160 \approx 2^{90.6}$.

A.3 Replacing the inner S-boxes: W-cache and UU-cache

After §A.2 the per-ω cost is $c_\omega^{\text{on}} = 3.19$ and $c_\omega^{\text{off}} = 2.06$. We now reduce both to ≈1.

Online: the W-cache. Per (pair, k_{-1}-guess), precompute

$$W_r[v][j] = \text{InvMC}_{0,r} \cdot \text{InvSB}\big(C^{(v)}[p_r] \oplus U_r[j]\big)$$

for r=0..3, v=0..255, and j over the $|U_r|$ candidates for $u_6[p_r]$. The build costs $\sum_r |U_r| \cdot 255$ InvSB; it serves $\sum_\beta \prod_r \text{DDT}_r(\beta) \cdot 255$ entry-element pairs. With the AES DDT spectrum $\{0{:}129, 2{:}126, 4{:}1\}$:

$$c_{\text{iSB}}^{\text{on}} = \frac{E[\sum_r |U_r|]}{E[\#\text{cands}]} = \frac{4 \cdot (256/127)}{(256/127)^4} = 4 \cdot (127/256)^3 = 0.48837.$$

Verified on 160 000 real-AES pairs (4 seeds): 0.4884±0.0005. With the inverse folded into POWINV, $c_\omega^{\text{on}} = 1 + 0.488 = \mathbf{1.49}$.

Offline: the UU-cache. The XOR-separability extends one level up: the 20 DDT-branch bits partition into four disjoint groups G_r (one z_4-bit + four x_3-bits per anti-diagonal r), and

$$d_\omega(20 \text{ bits}) = \bigoplus_{r=0}^{3} \text{UU}_r[g_r][\omega], \qquad g_r \in [0, 32).$$

Precompute 4×32 tables once per 10-parameter base (≈160 S-boxes); serve 2^{20} entries at 4 reads + 3 XORs each. Amortized $c_{\text{SB}}^{\text{off}} = 160/2^{20} \approx 1.5{\times}10^{-4}$; $c_\omega^{\text{off}} = \mathbf{1.00}$. *Verified* 62 914 560/62 914 560 against cold recomputation.

Cost. Online c_ω=1.49 gives $T = 2^{88}(255{\cdot}1.49 + 15)/160 = \mathbf{2^{89.30}}$. Offline c_ω=1.00 gives $P = 2^{88.75} \le T$. The fingerprint combine is now the limiting term; §A.5 reduces C_{post} from 15 to 8.

A.4 Collision entropy of the Add-0 fingerprint

We repeat the §5.1.1 experiment on the 256-element Add-0 multiset of §4.1, again at $N = 3 \times 10^9$. Unlike the raw fingerprint, the Add-0 $I_{m,n}$ fingerprint is *not* full-entropy:

multiset	k=1	k=2	k=3	k=5	k=6	k=7	k=8
255 (raw)	7.994	15.99	23.98	39.97	47.97	56.2	—
256 (Add-0)	7.994	15.01	20.84	36.83	44.75	**45.02**	53.03
ideal $k \log_2 255$	7.994	15.99	23.98	39.97	47.97	55.96	63.95

The gap to ideal is ≈ 1 bit at k=2 and settles at ≈ 3.1 bits over $k = 3..6$; but at k=7 it jumps to ≈ 10.9 bits and stays there (the k=7 point rests on 1.3×10^5 collisions, $\sigma = 0.004$ bits, so this is structure, not noise): byte 7 of the fingerprint is essentially determined by bytes 1–6. Both dependencies have the same source. At even multiset size every Lucas term of a weight-3 exponent carries a factor of S_1 (§4.1), so the five sums $P_7, P_{11}, P_{13}, P_{19}, P_{37}$ share that factor and the ratios built from them are algebraically related. Extrapolating the slope beyond k=8 (assuming no further loss in bytes 9–12, which we have not measured), the 12-byte Add-0 key carries $\approx 96 - 10.9 \approx 85$ bits — *below* the ≈ 91 bits the false-positive budget of §4.1 requires — whereas the odd-parity key carries the full ≈ 95.9.

This defect is confined to the $I_{m,n}$ fingerprint. It is one of the reasons the final complexity (§4.1.1) and both end-to-end experiments (§5.3, §5.4) use the $\chi^\star$ fingerprint instead: measured the same way, the stored $\chi^\star$ key follows a straight line of slope 7.79 bits/byte through k=7 at both parities (§5.1.2, Figure 3), with no analogous cliff.

A.5 Faster canonicalization ($\chi^{\star\prime}$)

The $(\alpha^\star, \beta^\star)$-solve of §4.2 costs 11 lookups. Since $\alpha^\star$ depends only on (S_1, S_3, S_5, S_7) through P_7, and $\beta^\star$ only on (S_1, S_3), we precompute a 256^2-entry table $(\alpha^\star, \beta^\star) \leftarrow (\log P_7, S_3/S_1^3)$, reducing the solve to 3 antilogs and one 2-byte lookup: $C_{\text{post}} = \mathbf{8}$. This gives $\ell = 255{\cdot}1.49 + 8 = 388$ and $T = 2^{89.28}$ — a $\approx$0.03-bit gain, since C_{post} is $\sim$2% of ℓ and the $255\,c_\omega$ term dominates. (The offline phase, with $c_\omega^{\text{off}} = 1.00$, reaches $\ell = 263$ and $P = 2^{88.72} \le T$; the online $c_\omega = 1.49$ remains the binding cost.) The 128 KB table is negligible against M.

B Degenerate-case accounting

Every special branch of the fingerprint and bridge pipeline, with probability (analytic; two independent implementations agree at 10^7–10^8 samples), handling rule, and amortized cost. Branches B* are bridge-side; R*/A* are $\chi^\star$ fingerprint-side at odd / even (Add0) parity respectively.

id	condition	probability	handling / cost
B0	s=0 (ref. value zero)	2^{-8}	δ-set discarded; $\times 256/255$ on D
B1/B1′	bad-index parity odd / even	$\approx 0.491/0.509$	Add0 vs. raw fingerprint; two lookups
R0	$P_7 \neq 0$	$1-2^{-8}$	generic; $\alpha^\star = P_7^{-1/7}$
R1–R2	fallback to P_{11}, P_{13}	$2^{-8}, 2^{-16}$	+2 lookups each
R3	$P_7{=}P_{11}{=}P_{13}{=}0$	2^{-24}	extended fallback (see A4)
A0	$S_1 \neq 0,\ P_7 \neq 0$	$\approx 1-2^{-7}$	generic
A1–A3	as R1–R3, $S_1 \neq 0$	$2^{-8..-24}$	same handling
A4	$\mathbf{S_1{=}0}$	$\mathbf{2^{-8}}$	$\alpha^\star$: $P_{23}{\to}P_{31}{\to}P_{127}$; $\beta^\star$: min over
all 256 translations;	(all wt-3 P_m vanish together)		**amortized** <0.05 **lookups/entry**

With the handling above, the amortized contribution to T is <0.001 bits. The bridge-consistency cost (Algorithm 2, $\approx 2^{84}$ encryptions) is the only sub-dominant term worth listing.

C The balanced variant: $\max(D, T, M) = 2^{96.3}$

The main attack uses a single offline table and tries many pairs of plaintexts until the difference of one pair follows the differential of the table; this needs $D = 2^{105}$ chosen plaintexts. But we can also build such a table for the same differential placed at other byte positions of the state, and for one further variant of the differential. This reduces the data complexity at a cost of increased memory and runtime, but can be worth it because currently the data complexity dominates our attack.

This balanced variant builds 480 tables in total; any given pair then has 480 chances to match a table instead of one, reducing the data complexity down to $D = 2^{96.1}$ chosen plaintexts. The extra cost is building the 480 tables; it stays close to 2^{96} because each table is indexed by an ordered fingerprint that reads only 16 of the 255 δ-differences of an entry. The result is $(D, T, M) = (2^{96.1}, 2^{96.3}, 2^{96.1})$, derived step by step below. It rests on one fingerprint assumption, stated next; counting false positives instead, as Derbez, Fouque, and Jean do, gives $2^{96.9}$ (last paragraph), and the trail-probability convention shared with [DFJ13] moves D by up to 0.2 bit.

Assumptions and verification. As in Derbez, Fouque, and Jean [DFJ13], trail probabilities are estimated in the Markov model, which cannot be verified at these magnitudes; and a wrong candidate is taken to match a stored fingerprint at the random rate of 2^{-8} per fingerprint byte, which we verified on the real cipher down to pairwise match probabilities of 2^{-62} (the 2^{-96} rate itself is beyond reach), and which the last paragraph removes by a counting argument. The coverage of the fingerprint and the equality of the online and offline fingerprints on right pairs are measured on $3 \cdot 10^8$ right pairs of the real cipher, and the per-entry cost is confirmed by an independent implementation; the enumeration of the 480 tables is counted, not run.

The data and the tables. The main attack collects its pairs from structures of 2^{32} plaintexts: the four bytes of the diagonal $\{0, 5, 10, 15\}$ take all values and the other twelve bytes are fixed. The balanced variant queries one larger structure of 2^{96} plaintexts: the twelve bytes of the three diagonals $\{0, 5, 10, 15\}$, $\{1, 6, 11, 12\}$ and $\{2, 7, 8, 13\}$ take all values and the fourth diagonal $\{3, 4, 9, 14\}$ is fixed. All pairs of the attack come from this one structure, and every δ-set is built inside it, exactly as in the main attack, from one member of a candidate pair and the guessed k_{-1} bytes of one diagonal.

The 480 tables cover two shapes of the differential, both as used in [DFJ13, §3.3] for their own data–memory trade-off and with the fingerprint designed in this work. The first

has two active plaintext diagonals and one active byte of x_5; a placement is a choice of the two diagonals among the three ($\binom{3}{2} = 3$), of the diagonal of x_1 that carries its two active bytes (4), and of the active byte of x_5 (16), so there are $3 \cdot 4 \cdot 16 = 192$ of them, and we call them the $(2,1)$ tables. The second has one active plaintext diagonal and two active bytes of x_5 in the same column; this two-ended shape is sketched in [DFJ13, §3.3]; a placement is the diagonal (3), the active byte of x_1 (4), and the column and pair of bytes of x_5 ($4 \cdot \binom{4}{2} = 24$), so there are $3 \cdot 4 \cdot 24 = 288$ of them, and we call them the $(1,2)$ tables. Every table has 2^{88} entries, one per value of the eleven offline parameter bytes, as in the main attack.

The online key bytes. For a candidate pair the attacker enumerates nine key bytes: the four bytes k_{-1} of the active plaintext diagonal, the four bytes u_6 of the anti-diagonal that leads to the active bytes of x_5, and one more byte $k_0[\beta]$:

$$K_{\text{on}} \;=\; k_{-1}[\text{diagonal}]\ (4) \;\cup\; u_6[\text{anti-diagonal}]\ (4) \;\cup\; k_0[\beta]\ (1), \qquad |K_{\text{on}}| = 9.$$

So this variant has $|K_{\text{on}}| = 9$, not 8: the Möbius bridge of §4.1 still removes the value byte $u_5[\rho]$ from the guess, but the ordered fingerprint below needs the index byte $\xi := x_1^{(0)}[\beta]$, and $k_0[\beta]$ is the key byte that gives it; the freed byte is re-spent rather than saved.

The ξ-ordered fingerprint. Fix one table entry. Its 255 δ-differences at the match byte ρ of x_5 can be labeled by the post S-box difference $\eta = \Delta z_1[\beta]$, which the table computes from its own parameters; write d_η for the difference with label η. The entry keeps only the first fourteen labels, sets $D_j := d_j^{-1}$ for $j = 1, \ldots, 14$, and stores the twelve ratio bytes

$$\Phi(D_1, \ldots, D_{14}) \;=\; \left(\tfrac{D_3 \oplus D_1}{D_2 \oplus D_1},\ \tfrac{D_4 \oplus D_1}{D_2 \oplus D_1},\ \ldots,\ \tfrac{D_{14} \oplus D_1}{D_2 \oplus D_1}\right) \in \mathrm{GF}(2^8)^{12},$$

with a fixed relabeling rule when $D_2 = D_1$, applied on both sides.

This is a valid index for three reasons. First, Φ does not change when every D_j is replaced by $\alpha D_j \oplus \beta$ with $\alpha \neq 0$, because such a map cancels in every ratio; so no canonicalization over (α, β) is needed, and this is what makes an entry cheap. Second, online, after peeling the last two rounds with the guessed bytes, the attacker holds values g_v labeled by the pre-S-box difference $v = \Delta x_1[\beta]$, and the Möbius bridge of §4.1 gives $g_v = s^2 D_{\eta(v)} \oplus s$ with one common pair (s^2, s) for all labels; this is a map of the kind just described, so the online fingerprint equals the offline one on a right pair. Third, the online labels are v and the offline labels are η, and they correspond by the permutation $\eta = \sigma_\xi(v) = S(\xi \oplus v) \oplus S(\xi)$; so the online side needs the byte ξ, and this is exactly what the ninth key byte $k_0[\beta]$ supplies. This is why the argument of §4.1 that a fingerprint must read all 255 values does not apply here: that argument is about a fingerprint that has to work for every unknown ξ, while the guessed $k_0[\beta]$ fixes ξ and with it the matching of the labels on the two sides; the ninth byte is the price of reading only 16 of them.

Reading an entry costs $W_{\text{off}} \approx 42$ lookups: one Gray-walk step over the 14 labels, one inversion per label, and the 12 ratio bytes. The τ-variant used below stores 16 labels and lets the enumeration of the table probe the stored online fingerprints with 15 probes per entry, for $W_{\text{off}}^{\tau} \approx 64$.

The bridge relation fails at a label whose δ-difference is 0 or equals s, two values in 256, and the whole variant fails when $s = 0$; so a right pair is detected with probability

$$\eta_{\text{cov}} \;=\; \left(\tfrac{254}{256}\right)^{14} \tfrac{255}{256} \;\approx\; 0.893,$$

and we measured 0.892491 ± 0.000018 on $3 \cdot 10^8$ right pairs of the real cipher, the online and offline fingerprints agreeing on every exception-free instance. This loss belongs to the bridge; the attack of Derbez, Fouque, and Jean, which guesses $u_5[\rho]$ and matches the

difference sequence directly, does not have it. The τ-variant stores 16 labels and matches on any of the 15 windows of 14 labels obtained by deleting one pair of adjacent labels from the 16 (the first of these windows is the clean one of the first 14 labels); it is stopped only by $s = 0$ or by a set of bad labels that is not empty, one label, or one adjacent pair, so its coverage is

$$\eta^{\tau}_{\mathrm{cov}} \;=\; \tfrac{255}{256}\big(\tfrac{254}{256}\big)^{14}\Big(1 + 14\cdot\tfrac{2}{256}\Big) \;\approx\; 0.990,$$

measured 0.990130 ± 0.000006 on the same $3\cdot 10^8$ right pairs, again with the two fingerprints agreeing on every covered instance.

Data. Write $q := 2^{32}$ for the number of values of one diagonal, so the structure B has $|B| = q^3 = 2^{96}$ plaintexts. A pair of B differs on one, two or three of the three active diagonals; fixing which a diagonals differ, there are $\binom{3}{a}(q-1)^a q^3/2$ such unordered pairs, so the numbers of pairs at diagonal distance a are

$$\mathrm{pd}_1(B) = 3(q-1)\,\tfrac{q^3}{2} \approx 2^{128.58}, \qquad \mathrm{pd}_2(B) = 3(q-1)^2\,\tfrac{q^3}{2} \approx 2^{160.58},$$

and every such pair has its δ-set inside B and can be used. Let $p_{a,b}$ be the probability that a distance-a pair follows the (a,b) differential, with b active bytes of x_5 in one column, summed over the $16\binom{4}{b}$ positions. The differential imposes $3a$ byte conditions at x_1, 12 at x_4 and $4-b$ at x_5, each passed with probability 2^{-8} in the convention of [DFJ13], so

$$p_{a,b} \;=\; 16\binom{4}{b}\,2^{-8(3a+16-b)}, \qquad p_{1,2} = 2^{-129.42}, \quad p_{2,1} = 2^{-162.0}, \quad p_{1,1} = 2^{-138.0}.$$

The expected number of right pairs in the structure is the pair count times the probability, class by class:

$$\begin{aligned} E(B) &= \underbrace{p_{1,2}\,\mathrm{pd}_1(B)}_{(1,2)} + \underbrace{p_{2,1}\,\mathrm{pd}_2(B)}_{(2,1)} + \underbrace{p_{1,1}\,\mathrm{pd}_1(B)}_{(1,1)} \\ &= 2^{-0.84} + 2^{-1.42} + 2^{-9.42} \;\approx\; 0.56 + 0.37 + 0.0015 \;=\; 0.94. \end{aligned} \tag{5}$$

With the coverage $\eta^{\tau}_{\mathrm{cov}} \approx 0.99$ the structure carries 0.93 detected right pairs, and one expected right pair ($P_{\mathrm{succ}} \approx 1 - 1/e$) needs $D = 2^{96}/0.93 = 2^{96.1}$ chosen plaintexts, the extra 7% being taken from part of one value of the fourth diagonal. (Pricing a byte condition at $\frac{255}{256}$ or $\frac{256}{255}$ instead of 2^{-8} moves D by less than 0.1 bit.)

Cost. The offline work is the enumeration of the 480 tables of 2^{88} entries each (the $(1,1)$ tables contribute 0.2% of E and are dropped):

$$\begin{aligned} N_{\mathrm{off}} &\;=\; 480\cdot 2^{88} \;=\; 2^{96.91} \text{ entries}, \\ T_{\mathrm{off}} &\;=\; N_{\mathrm{off}}\, W^{\tau}_{\mathrm{off}}/160 \;=\; 2^{96.91}\cdot 64/160 \;=\; 2^{95.59}. \end{aligned}$$

Collecting the pairs means hashing the $2^{96.1}$ ciphertexts by their inactive bytes, at ≈ 30 operations per text (operation count). A $(2,1)$ table needs a pair that differs on its two active diagonals, a distance-2 pair, whose inactive 12 ciphertext bytes off the active anti-diagonal must agree: this is a 96-bit filter, and there are 4 anti-diagonals. A $(1,2)$ table needs a distance-1 pair whose inactive 8 bytes off a pair of anti-diagonals must agree: a 64-bit filter, and there are $\binom{4}{2} = 6$ pairs of anti-diagonals. So the number of survivors is

$$\underbrace{4\cdot 2^{-96}\mathrm{pd}_2(B)}_{(2,1)\text{: } 2^{66.6}} \;+\; \underbrace{6\cdot 2^{-64}\mathrm{pd}_1(B)}_{(1,2)\text{: } 2^{67.2}} \;\approx\; 2^{67.9}.$$

Table 7: Cost of the balanced variant, in the 160-S-box convention. N_{off} and N_{on} count table entries and fingerprints; only the T-rows are in encryptions, each entry or fingerprint costing a fraction of one.

quantity	derivation	value	unit
D	one structure, plus 7% of a fourth diagonal value	$2^{96.1}$	chosen plaintexts
N_{off}	480 tables $\times\, 2^{88}$ entries	$2^{96.91}$	entries computed
T_{off}	$N_{\text{off}} \cdot 64/160$	$2^{95.59}$	encryptions
N_{on}	$2^{67.9} \cdot 2^{20.03} \cdot 2^{8}$	$2^{95.9}$	fingerprints
T_{on}	$N_{\text{on}} \cdot 56/160$	$2^{94.4}$	encryptions
T_{pf}	$2^{96.1} \cdot 30/160$	$2^{93.68}$	encryptions
T_{fp}	$2^{91.8}$ first hits at 12 lookups, $2^{84.8}$ rebuilds at 2^{11}	$\approx 2^{89.3}$	encryptions
T	$T_{\text{off}} + T_{\text{on}} + T_{\text{pf}} + T_{\text{fp}}$	$2^{96.3}$	encryptions
M	$2^{96.1}$ ciphertexts + one $2^{88.5}$-block table + 2^{91} blocks of stored fingerprints	$2^{96.1}$	128-bit blocks
P_{succ}	$1 - e^{-\eta^{\tau}_{\text{cov}} E(D)}$, $\eta^{\tau}_{\text{cov}} E(D) = 1$	$1 - 1/e$	

A survivor fixes the active diagonal, anti-diagonal or anti-diagonals, but not the active byte of x_1 nor the active bytes of x_5, so it is compatible with up to $4 \cdot 4 = 16$ tables; for each of them the input side of the DDT leaves $\approx 2^8$ values of $k_{-1}[\text{diagonal}]$ and the output side $\approx 2^8$ values of $u_6[\text{anti-diagonal}]$, exactly as the 2^{16} candidates per pair of the main attack, so a survivor yields $16 \cdot 2^8 \cdot 2^8 = 2^{20}$ triples $(\text{table}, k_{-1}[\text{diagonal}], u_6[\text{anti-diagonal}])$ ($2^{20.03}$ measured), and each triple is completed by the 2^8 values of $k_0[\beta]$; one fingerprint costs $\ell_{\text{on}} \approx 48$ lookups, 56 with the 16 labels and the two extra ratio bytes of the τ-variant:

$$\begin{aligned} T_{\text{pf}} &= 2^{96.1} \cdot 30/160 = 2^{93.68}, \\ N_{\text{on}} &= 2^{67.9} \cdot 2^{20.03} \cdot 2^{8} = 2^{95.9} \text{ fingerprints}, \\ T_{\text{on}} &= 2^{95.9} \cdot 56/160 = 2^{94.4} \qquad (2^{94.2} \text{ at } \ell_{\text{on}} = 48). \end{aligned}$$

The matching runs table by table, in the opposite direction to the main attack: the online fingerprints of the candidates whose survivor is compatible with the table in scope, $2^{95.9}/480 \approx 2^{87}$ of them, are stored, keyed by each of their 15 windows, and the enumeration of the 2^{88} entries of that table probes them, 15 probes per entry; the table is then discarded and the next one built. A window key is the 12 ratio bytes, a 2^{-9} filter against the 2^{87} stored keys of that window, so the $480 \cdot 2^{88} \cdot 15 = 2^{100.8}$ probes return $\approx 2^{91.8}$ first hits. Each stored fingerprint also carries the two ratio bytes of the labels 17 and 18, and a first hit is kept only if the entry, walking two labels further, agrees with one of them; this costs ≈ 12 lookups per hit and passes 2^{-7} of the wrong ones, so the $\approx 2^{84.8}$ survivors are rejected by rebuilding the entry's full sequence at $\approx 2^{11}$ lookups each (operation count), and $T_{\text{fp}} \approx 2^{91.8} \cdot 12/160 + 2^{84.8} \cdot 2^{11}/160 \approx 2^{89.3}$; a right pair is lost at this step only when both extra labels are bad, $(2/256)^2$, which does not change η^{τ}_{cov}. Memory holds the $2^{96.1}$ ciphertexts, one $2^{88.5}$-block table at a time, and the $\approx 15 \cdot 2^{87}$ stored window keys of that table ($\approx 2^{91}$ blocks), so $M = 2^{96.1}$. Table 7 collects the terms.

Total. Summing the time rows of Table 7, $T = 2^{95.59} + 2^{94.4} + 2^{93.68} + 2^{89.3} = 2^{96.3}$; so $(D, T, M) = (2^{96.1}, 2^{96.3}, 2^{96.1})$, and $\max(D, T, M) = 2^{96.3}$ at $P_{\text{succ}} \approx 1 - 1/e$. This is $\log_2(2^{99}/2^{96.3}) \approx 2.7$ bits below the balanced point $(D, T, M) = (2^{97}, 2^{99}, 2^{98})$ of [DFJ13], under the same one-expected-right-pair convention.

Where the gain comes from. The single structure and the 480 placements lower the data, but by themselves they do not lower the time: the gain is in the cost of a table entry, and Table 8 prices the same 480 tables with three kinds of entry. With the multiset entries

Table 8: The same 480 tables priced with three kinds of entry; D and M are $2^{96.1}$ in every row, the data of [DFJ13]'s balanced point is 2^{97}.

entry type	$\lvert K_{\text{on}}\rvert$	T_{off}	T_{on}	$\max(D, T, M)$
[DFJ13] balanced point, as published	9			2^{99}
same 480 tables, their multiset entries	9	$2^{98.6}$	$2^{97.7}$	$2^{99.2}$
same 480 tables, our $\chi^\star$ fingerprint	8	$2^{97.6}$	$2^{89.2}$	$2^{97.6}$
same 480 tables, the ordered fingerprint (ours)	9	$2^{95.6}$	$2^{94.4}$	$2^{96.3}$

of Derbez, Fouque, and Jean (the full 255-element sequence per entry, and their online loop over $u_5[\rho]$), the 480 tables cost $2^{99.2}$, no better than their own balanced point: the layout alone gains nothing. With our $\chi^\star$ fingerprint of §4.2 (all 255 differences, $\lvert K_{\text{on}}\rvert = 8$) they cost $2^{97.6}$. With the ordered fingerprint (16 of the 255 differences, $\lvert K_{\text{on}}\rvert = 9$) they cost $2^{96.3}$. So all of the 2.7 bits below [DFJ13] comes through the Möbius bridge, which is what makes both cheap fingerprints usable; the ninth key byte is the price of the ordered one.

Without the fingerprint assumption. We can drop the 2^{-8}-per-byte assumption and count false positives as Derbez, Fouque, and Jean do, at the price of a longer fingerprint. If each entry keeps the first 30 labels and the windows delete one adjacent pair of them, a window key is 26 ratio bytes, and the $2^{101.8}$ probes of the whole attack meet the 2^{87} stored keys of a window in a space of 2^{208} values: the expected number of false matches is $2^{101.8+87-208} = 2^{-19}$, so no false match is expected at all, and what remains is only the heuristic, shared with [DFJ13], that a wrong candidate gives a random-looking sequence. An entry then costs $W^{\tau}_{\text{off}} \approx 30 \cdot 2 + 28 + 29 = 117$ lookups, the coverage of the 30 labels is 0.975, $T_{\text{off}} = 2^{96.91} \cdot 117/160 = 2^{96.46}$, the false-positive term disappears, and

$$(D, T, M) = (2^{96.1}, 2^{96.9}, 2^{96.1}), \qquad \max(D, T, M) = 2^{96.9} \text{ at } P_{\text{succ}} \approx 1 - 1/e,$$

about 2.1 bits below the balanced point of [DFJ13].

D Wallclock runtime evaluations

Throughout this paper we argue that the attack we have developed is more efficient than that of prior work by analyzing the runtime and counting AES S-box evaluations. In this section we estimate it numerically, by timing the attack on actual hardware.

Even reduced round AES-128 attack cannot be run with the hardware commercially available within reasonable time. Instead, we can run the same attack on a cipher small enough to finish and compare its wall-clock time with the attack of Derbez, Fouque, and Jean. And we can measure, on the real (reduced round) AES-128, the cost of one offline table entry and the cost of one online candidate, the two numbers that are multiplied by 2^{88} and 2^{88} in the count. Then, because the two attacks share the same optimizations, any remaining difference between them is due to the Möbius bridge.

A complete attack on a small-scale AES. $\mathrm{SR}(7, 2, 2, 6)$ is a 7-round AES with a 2×2 state of 6-bit words, so the key has 24 bits and the offline table has 71,282,781,952 entries; we built the full table once and ran the complete key recovery, black-box, on 50 random keys for our attack and for the attack of Derbez, Fouque, and Jean with our table-enumeration optimizations. Both recover 50/50 keys. Our online phase makes 64× fewer table lookups ($2.15 \cdot 10^7$ against $1.37 \cdot 10^9$ per key), which is the word size 2^6 of this cipher, the analogue of the 2^8 of AES. In wall-clock time the ratio is 18 to 20× (38 s against 680 s per key):

Table 9: Black-box key recovery on $\mathrm{SR}(7, 2, 2, 6)$, the same 50 keys for all three; all three recover all 50 keys. All numbers are means over the 50 keys, with the wall-clock medians in parentheses. The first two columns share our table enumeration and code base; the third is the attack of Derbez, Fouque, and Jean as published. Key recovery per key includes collecting the data, the online phase, the rejection of false positives, and the trial encryption; the full runtime adds the table build amortized over the 50 keys. The timing and lookup rows ran with a 2^{41}-bit Bloom filter in front of the table, its false positives rejected by the key-schedule test and counted in the time; the last row is measured on an exact-fingerprint table with no Bloom filter, and counts genuine fingerprint collisions only. All three on the same machine.

	ours	[DFJ13] with our optimizations	[DFJ13] as published
table build, wall clock (80 threads)	27 min	14 min	16 min
table lookups per key	$2.15 \cdot 10^7$	$1.37 \cdot 10^9$	$1.37 \cdot 10^9$
key recovery, wall clock per key	38 s (36)	680 s (517)	757 s (571)
full runtime per key (build/50 + recovery)	71 s	696 s	776 s
chosen plaintexts per key	$1.08 \cdot 10^4$	$1.08 \cdot 10^4$	$1.08 \cdot 10^4$
fingerprint collisions per lookup (exact table, no Bloom filter)	$1.74 \cdot 10^{-4}$	$4.4 \cdot 10^{-9}$	$4.4 \cdot 10^{-9}$

our lookup is more expensive than theirs, because it computes the $\chi^\star$ fingerprint of §4.2 where theirs hashes a multiset, and on a 64-element δ-set this overhead is a factor ≈ 2. We also ran their attack as published, with the table built by cold per-entry enumeration and the ciphertext peel recomputed for every value of $u_5[0]$ in the online loop; this changes only the time, not the lookups or the data. On this small cipher the published variant is only 1.1× slower than the optimized one, because the state is 2×2 and the peel is a small part of the loop; on AES-128 the same comparison is 2.8× per online candidate and 40× per table entry (see below), so the small cipher understates this third column. Table 9 collects the numbers; the timed runs keep the table behind a 2^{41}-bit Bloom filter, and a second run with the exact fingerprints stored measures the genuine collision rate per lookup, $1.74 \cdot 10^{-4}$ (ours) and $4.4 \cdot 10^{-9}$ (theirs), all false hits correctly rejected.

One offline entry of the AES-128 attack. We generated genuine table entries of the 7-round AES-128 attack (real offline parameters, the full DDT-Gray walk of §A.2, the 255-element sequence of δ-differences, the complete $\chi^\star$ fingerprint) and timed three ways of producing one entry on the same core. Ours costs 376 cycles per entry at 1.07 S-box lookups per element (the walk itself is ≈ 29 cycles, the fingerprint ≈ 326). The attack of Derbez, Fouque, and Jean with the same walk and a multiset hash instead of the fingerprint costs 287 cycles. Their attack as published, with each entry traced cold through rounds 2–4, costs 11,493 cycles at 24.1 S-box lookups per element. A partial table of 2^{36} genuine entries built this way contains no duplicate entries and no deviation of its 64-bit words from uniform, and 2^{34} genuine wrong candidates probing it produce no fingerprint hit.

One online candidate of the AES-128 attack. The comparable unit of the online phase is one guess of the diagonal bytes of k_{-1} and the anti-diagonal bytes of u_6: for us this is one fingerprint, for Derbez, Fouque, and Jean it is their inner loop over the 256 values of $u_5[0]$. On real 7-round ciphertexts, fully vectorized with GFNI and AVX-512 on both sides, our candidate costs 649 cycles and theirs 117,141; in clock-neutral interleaved runs the ratio is 277×, range 224 to 341, which is the removed byte 2^8 seen in wall clock. Against their inner loop as published, with the ciphertext peel recomputed for every value of $u_5[0]$, the ratio is 689×, range 510 to 777. Every kernel in these runs is checked bit for bit: our online fingerprint equals the fingerprint of the true internal sequence of the cipher for the correct key, and the multiset hash of their loop equals it for the correct $u_5[0]$.

Table 10: Full-attack time at $D = 2^{105}$: the measured cost of one table entry and of one online candidate, multiplied by the counted number of entries and candidates. A candidate of [DFJ13] contains the 256-value loop over $u_5[0]$. Nothing here runs more than 2^{26} operations.

	ours	[DFJ13] with our optimizations	[DFJ13] as published	unit
offline cost	376	287	11,493	cycles per entry
online cost	649	117,141	371,236	cycles per candidate
table entries	2^{88}	2^{80}	2^{80}	entries (counted)
online candidates	2^{88}	2^{88}	2^{88}	candidates (counted)
total	$2^{98.0}$	$2^{104.84}$	$2^{106.50}$	cycles

From the constants to the full attack. Multiplying the measured constants by the counted exponents at $D = 2^{105}$ gives the time of Table 10: 2^{88} entries and 2^{88} candidates for us, 2^{80} entries and 2^{88} candidates, each with the 256-value loop, for Derbez, Fouque, and Jean. In cycles the totals are $2^{98.0}$ for us against $2^{104.84}$ with our optimizations and $2^{106.50}$ as published, so the measured gains are $2^{6.84}$ and $2^{8.50}$. The counted ledger of Table 2 gives 2^8 and $2^{9.72}$ for the same two comparisons; the measured figures are about one bit lower, because our measured online kernel does not include the W-cache of §A.3 ($\ell_{\text{on}} = 1538$ counted lookups instead of 388), and because our fingerprint runs at 0.42 cycles per counted lookup while the published loop runs at 0.95. In units of a software T-table encryption (195.7 cycles on the same machine) our total is $2^{90.4}$, close to the counted $2^{89.28}$; in units of an AES-NI encryption (5.63 cycles) it is $2^{95.5}$, which is what the same work costs when brute force is priced at AES-NI speed.