

Polytopic Sieving: Practical Low-Data Key-Recovery Attacks on Reduced-Round AES

Stefan Kölbl 

Google, Switzerland
kste@google.com

Abstract. Minimizing the data complexity required to recover the secret key of reduced-round block ciphers is a fundamental problem in symmetric cryptanalysis. Here, we introduce Polytopic Sieving, a data-efficient key-recovery framework and apply it to reduced-round AES. By characterizing the algebraic dependencies of anchor bytes (the reference state values that govern differential transitions across S-boxes), we show that cross-column and cross-row geometric consistency substantially restricts the realizable subkey space.

We apply this framework to 4-round AES to achieve a practical key recovery using only 3 chosen plaintexts in 2^{40} time complexity or 4 chosen plaintexts in less than 2^{34} time complexity. This sets a new benchmark for data efficiency of polytopic attacks, and outperforms other recent techniques such as Subspace Trail Cryptanalysis and Mixture-Integral attacks on very low data targets. Furthermore, we extend our framework to a 5-round attack which requires only 10 plaintexts. By pairing our sieving with a dissected meet-in-the-middle approach, we can reduce both the data and time complexity over the previously best known polytopic attacks. These results establish the lowest data requirements known to date for practical key recovery on 4-round AES.

Keywords: AES, Polytopic Cryptanalysis

1 Introduction

Standardized in 2001, the Advanced Encryption Standard (AES) [AES01] has served as the benchmark for symmetric cryptography, undergoing decades of intense scrutiny. Nonetheless, cryptanalysts continue to push the boundaries of AES security, with several novel techniques developed in recent years that steadily advance attacks on round-reduced variants.

In practical adversarial settings, an attacker’s data collection capabilities are frequently constrained. Consequently, a fundamental theoretical and practical problem in symmetric cryptanalysis is: *What is the minimal data requirement to recover the secret key of reduced-round AES, and how computationally efficient can such ultra-low-data attacks be?* We will focus on this scenario in this paper and improve the state-of-the-art of low-data attacks on AES.

Differential cryptanalysis [BS91] evaluates pairs of plaintexts and ciphertexts, and has been one of the fundamental techniques to analyze block ciphers. However, differential attacks are typically of a statistical nature and the probability of multi-round differential characteristics is low, demanding substantial data volumes to construct a useful distinguisher. The AES also comes with strong security bounds against this technique. In this paper we revisit polytopic cryptanalysis, which extends this from simple pairs to a set of plaintexts and ciphertexts with their corresponding differences.

Our Contribution

We present the following contributions:

- **Polytopic Sieving Framework:** We formalize the algebraic relationship between reference state values (anchor bytes) and intermediate multi-difference trajectories across substitution-permutation networks. We show that enforcing joint geometric consistency across independent state columns filters candidate subkeys with high probability while requiring minimal data.
- **Practical Key Recovery on 4-Round AES:** We present key-recovery attacks against 4-round AES requiring only 3 chosen plaintexts in 2^{40} time and 2^{32} memory, or 4 chosen plaintexts in 2^{34} time and 2^{24} memory. This improves the time complexity of the previous best 3-plaintext attack by a factor of $2^{29.7}$ and enables key recovery on a single workstation.
- **Low-Data Attack on 5-Round AES:** We extend the framework to 5-round AES using 10 chosen plaintexts. By decomposing the 72-bit verification condition into four independent 16-bit functions and evaluating them via a 24-bit prefix meet-in-the-middle sieve, we achieve key recovery in $2^{61.7}$ 5-round encryptions and 2^{41} memory.
- **Experimental Validation:** We provide full implementations of the 4-round attacks on a workstation, validating the theoretical filtering probabilities and demonstrating end-to-end key recovery in practical time.

Related Work

There has been extensive cryptanalysis of reduced-round AES and we summarize the major cryptanalytic techniques related to our framework and contrast their mechanics. A full comparison with other low-data attacks is given in Table 1.

Classical differential cryptanalysis [BS91] tracks single input-output difference pairs through cipher rounds. Polytopic cryptanalysis, introduced by Tiessen [Tie16] pushed this concept to geometric structures (s -polytopes) consisting of $(d + 1)$ texts sharing simultaneous difference relations. Tiessen applied impossible polytopic transitions to 4-round and 5-round AES.

In parallel with statistical differentials, algebraic and guess-and-determine attacks treat the cipher as a structured system of non-linear equations over $\mathbb{F}_2$ s. Boullaguet et al. [BDF11] developed an automated constraint satisfaction framework to optimize guess-and-determine pathways on AES. Gilbert and

Minier [GM00] introduced collision attacks on 7-round Rijndael by identifying a functional property: the 3-round encryption function mapping an active input byte to an intermediate state byte can be fully parameterized by a small number of unknown constants. Similar algebraic dependencies were exploited in the Demirci-Selçuk Meet-in-the-Middle (DS-MitM) framework [DS08] on 8-round AES, which was later refined by Dunkelman, Keller, and Shamir [DKS10] via differential enumeration, and by Derbez, Fouque, and Jean [DFJ13] using automated graph search.

Grassi, Rechberger, and Rønjom [GRR16] introduced subspace trail cryptanalysis, which analyzes the propagation of affine subspaces through the AES round transformations. They proved that for certain input subspaces of dimension 4, the state after 3.5 rounds is guaranteed to belong to a specific coset of a subspace with probability 1. This yielded a 4-round key-recovery attack using 3 chosen plaintexts with a complexity of $2^{69.7}$ operations. Grassi and Schofnegger [GS20] subsequently formulated mixture-integral attacks, which combine integral cryptanalysis with subspace partitions, recovering the 4-round key using 6 chosen plaintexts in $2^{33.7}$ time. While mixture integrals achieve competitive time complexity, they require structured coset query sets that cannot be compressed below 6 plaintexts.

Meet-in-the-Middle attacks frequently face a bottleneck where the size of the intermediate meeting state exceeds the available memory. Canteaut, Naya-Plasencia, and Vayssière [CNV13] introduced the *Sieve-in-the-Middle* technique, which replaces direct table lookups with sequential sieving steps that test intermediate transitions against precomputed lookup tables of valid differential transitions. In parallel, Dinur, Dunkelman, Keller, and Shamir [DDKS12] introduced the *Dissection* technique for solving generalized knapsack and meet-in-the-middle problems with low memory overhead by recursively partitioning constraint equations. Our 5-round optimized attack synthesizes these two concepts in the multi-difference domain: we formulate an asymmetric 3-to-1 meet-in-the-middle decomposition over 72-bit constraint vectors and employ a 24-bit dynamic prefix filter.

2 Preliminaries

Let $\mathbb{F}_{2^8} \cong \mathbb{F}_2[x]/(x^8 + x^4 + x^3 + x + 1)$ denote the standard AES Galois field. The AES-128 block cipher operates on a 4×4 matrix of bytes (16 bytes = 128 bits) and applies four sequential transformations per round:

- **SubBytes** (S): An 8-bit non-linear S-box substitution applied independently to each byte. We define it over $\mathbb{F}_{2^8}$ as $S(z) = L(z^{-1}) \oplus c$, where $z \mapsto z^{-1}$ is Galois field inversion ($0^{-1} = 0$), L is an $\mathbb{F}_2$ -linear 8×8 bit-matrix, and $c = 0x63$ is the affine constant byte.
- **ShiftRows** (SR): A cyclic left-shift of row $r \in \{0, 1, 2, 3\}$ by r byte positions.
- **MixColumns** (MC): A linear diffusion layer operating independently on each 4-byte column via multiplication by the MDS matrix $M \in \mathbb{F}_{2^8}^{4 \times 4}$.
- **AddRoundKey** (ARK): Bitwise XOR with the 128-bit round subkey K^r .

Table 1. Comparison of low-data key-recovery attacks on AES for 4 and 5 rounds. Data complexity is measured in number of required known/chosen plaintexts (KP/CP). Time is given in r-Round AES encryption equivalents, while memory usage is given in plaintext (16-byte) equivalent.

Rounds	Attack	Data	Time	Memory
4	Guess-and-Determine [BDF11]	1 KP	2^{120}	2^{120}
4	Guess-and-Determine [BDF11]	2 CP	2^{80}	2^{80}
4	Diff. MitM [BDD ⁺ 10]	3 CP	2^{104}	1
4	Subspace Trail [GRR16]	3 CP	$2^{69.7}$	2^{12}
4	Polytopic Sieving (This Work)	3 CP	2^{40}	2^{32}
4	Polytopic Sieving (This Work)	4 CP	2^{34}	2^{24}
4	Guess-and-Determine [BDF11]	4 CP	2^{32}	2^{24}
4	Mixture Integral [GS20]	6 CP	$2^{33.7}$	2^{28}
4	Impossible Polytopic [Tie16]	8 CP	2^{38}	2^{15}
4	Subspace Trail [GRR16]	24 CP	$2^{35.1}$	2^{17}
5	MitM [Der13]	8 CP	2^{64}	2^{56}
5	Polytopic Sieving (This Work)	10 CP	2^{62}	2^{41}
5	Impossible Polytopic [Tie16]	15 CP	2^{70}	2^{41}

2.1 Notation

We address the bytes of the AES state matrix in the following column-major order:

$$\begin{pmatrix} s_0 & s_4 & s_8 & s_{12} \\ s_1 & s_5 & s_9 & s_{13} \\ s_2 & s_6 & s_{10} & s_{14} \\ s_3 & s_7 & s_{11} & s_{15} \end{pmatrix}. \quad (1)$$

Throughout this paper, we adopt the following notation to describe the internal states and differences of AES during encryption:

- SB^r : The 16-byte internal state at the beginning of round r (before **SubBytes**).
- SR^r : The internal state after applying **SubBytes** in round r .
- MC^r : The internal state after applying **ShiftRows** (and before **MixColumns**) in round r .
- AK^r : The internal state after applying **MixColumns** in round r .
- K^r : The 128-bit round subkey applied at the end of round r , such that $SB^{r+1} = AK^r \oplus K^r$.

Hence, one round of AES proceeds as

$$SB^r \xrightarrow{S} SR^r \xrightarrow{SR} MC^r \xrightarrow{MC} AK^r \xrightarrow{\oplus K^r} SB^{r+1} \quad (2)$$

We use the prefix Δ to denote the differential state corresponding to any internal state (e.g., ΔSB^r denotes the difference state before **SubBytes** in round r). When referring to a specific byte index $i \in \{0, \dots, 15\}$ of a state according to the matrix ordering, we use subscripts, such that X_i refers to byte s_i of state X .

2.2 Polytopes and Multi-Differences

Definition 1 (s -polytope). An s -polytope in $\mathbb{F}_2^n$ is an s -tuple of values in $\mathbb{F}_2^n$.

Definition 2 (d -difference). A d -difference over $\mathbb{F}_2^n$ is a d -tuple of values in $\mathbb{F}_2^n$ describing the relative positions of the texts of a $(d+1)$ -polytope from a single reference point.

For a $(d+1)$ -polytope $(m_0, m_1, \dots, m_d)$, we define the corresponding d -difference as:

$$(m_0 \oplus m_1, m_0 \oplus m_2, \dots, m_0 \oplus m_d).$$

We select the first text of the polytope (m_0) as the reference text and write the differences in the corresponding order of the remaining texts. We refer to the reference text m_0 as the *anchor* of the d -difference.

Example 1. Let (m_0, m_1, m_2, m_3) be a 4-polytope in $\mathbb{F}_2^n$. Then $(m_0 \oplus m_1, m_0 \oplus m_2, m_0 \oplus m_3)$ is the corresponding 3-difference with reference anchor m_0 .

The absolute state values of the anchor bytes at the inputs of active (non-zero difference) S-boxes play a central role in our analysis. We define standard variables for these active anchor bytes evaluated prior to the **SubBytes** layer across sequential rounds. We denote them by x_i , y_i , z_i , and w_i for active bytes entering round 1 (SB^1), round 2 (SB^2), round 3 (SB^3), and round 4 (SB^4), respectively.

3 Polytopic Sieving on 4-round AES

We use a 4-polytope as our initial structure. We encrypt $P = \{P_0, P_1, P_2, P_3\}$ with chosen input 3-differences $\Delta_{in} = (\delta_{in}^1, \delta_{in}^2, \delta_{in}^3)$, which are non-zero exclusively in the first byte position (s_0), to obtain the four ciphertexts $C = \{C_0, C_1, C_2, C_3\}$.

Our goal is to trace how this 4-polytope propagates through the round transformations up to the **MixColumns** layer in Round 3, and to analyze how the anchor bytes determine the reachable 3-differences observable at this state. We denote by ΔMC^3 the 3-difference tuple entering the **MixColumns** layer in Round 3 which will be the matching point for our attack.

Round 1. After key addition, we do not know the absolute values of the state, but the 3-difference Δ_{in} remains unchanged. Across the Round 1 S-box layer, the exact output 3-differences depend on the absolute values $P_0 \oplus K^0$. Crucially, these output differences depend solely on a single unknown anchor byte $x_0 = P_{0,0} \oplus K_0^0$. If we knew x_0 , we could fully determine the remaining polytope texts $P_{1,0} \oplus K_0^0$, $P_{2,0} \oplus K_0^0$, and $P_{3,0} \oplus K_0^0$, establishing the exact output 3-difference (see [Figure 1](#)). The subsequent **ShiftRows** transformation permutes the state bytes, and **MixColumns** diffuses the difference across the entire first column of $(AK_0^1, AK_1^1, AK_2^1, AK_3^1)$.

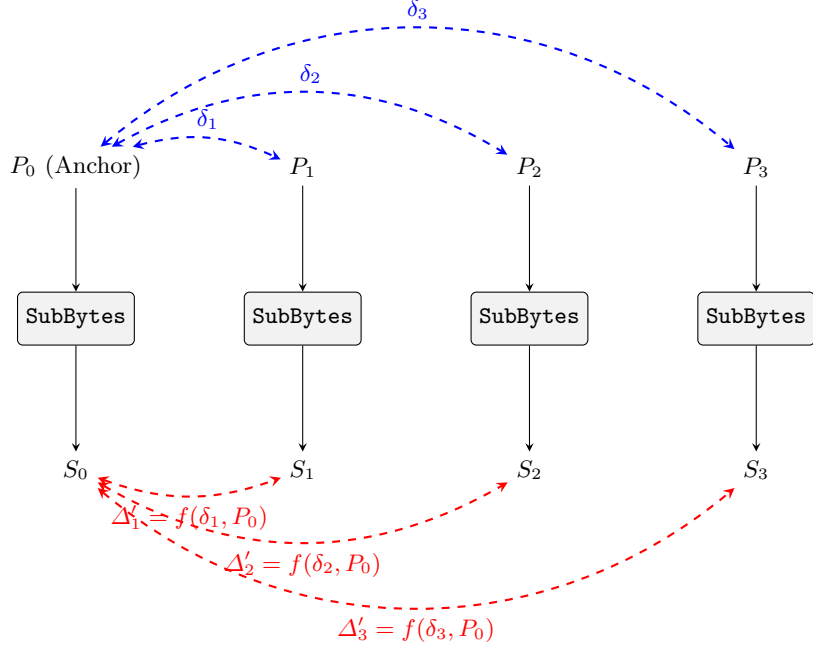


Fig. 1. Reference anchor bytes uniquely determine non-linear polytopic transitions. A known input 3-difference $(\delta_1, \delta_2, \delta_3)$ transitions through a non-linear layer into an unpredictable output 3-difference $(\Delta'_1, \Delta'_2, \Delta'_3)$. The exact output 3-difference is completely and uniquely determined by the absolute value of the specific reference anchor byte P_0 .

Round 2. The Round 1 output 3-difference $\Delta AK^1 = \Delta SB^2$ (where one column is active) propagates through the Round 2 S-box layer. Again, for each row $r \in \{0, 1, 2, 3\}$, a single unknown anchor byte y_r governs the S-box output difference. Knowing the four anchor bytes (y_0, y_1, y_2, y_3) entering Round 2 SubBytes fully determines the Round 2 S-box output differences. **ShiftRows** distributes these active bytes across all four columns, and **MixColumns** diffuses them so that the entire 16-byte state becomes active.

Round 3. The 3-difference ΔSB^3 now covers the full state matrix. To determine the exact difference propagation through the Round 3 S-box layer for the first column, each byte position again depends solely on a single unknown anchor byte z_r ($r \in \{0, 5, 10, 15\}$). Following **ShiftRows**, the observable 3-difference ΔMC^3 entering Round 3 **MixColumns** in the first column satisfies the functional dependency:

$$\Delta MC^3_{[0,1,2,3]} = \begin{pmatrix} f(\Delta_{in}, x_0, y_0, z_0) \\ f(\Delta_{in}, x_0, y_3, z_5) \\ f(\Delta_{in}, x_0, y_2, z_{10}) \\ f(\Delta_{in}, x_0, y_1, z_{15}) \end{pmatrix}. \quad (3)$$

A central property of Equation 3 is that ΔMC^3 depends strictly on these nine anchor bytes (x_0 , $y_{[0:3]}$, and $z_{[0,5,10,15]}$) and the initial 3-difference Δ_{in} , independent of the remaining round-key bytes. This observation was first made by Gilbert and Minier in [GM00]. In the original polytopic attack [Tie16], the same observation was used in the 5-round attack (just targeting the state after MixColumns).

The key observation here is that while these nine bytes span a 2^{72} search space, they exhibit a strong structural overlap, e.g., all four rows of ΔMC^3 depend on the same Round 1 anchor x_0 and not every row (before MixColumns) depends on all 9 bytes (see Figure 2). We will exploit these dependencies to construct an efficient key-recovery attack. Note that the polytope size ($s = 4$) is independent of the anchor dimension and represents the minimal choice required to make the subsequent filtering condition unique.

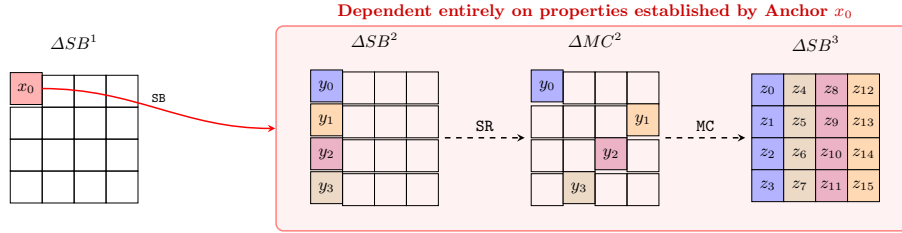


Fig. 2. Anchor state propagation and structural dependency overlap across rounds 1–3. The single initial anchor x_0 uniquely dictates the first algebraic transition. The subsequent four column anchors $y_0 \dots y_3$ are distributed across columns by ShiftRows. Entering Round 3, all 16 variables $z_0 \dots z_{15}$ become fully active, however a column solely depends on a single anchor x_0 and y_i at state ΔSB^3 .

3.1 Key Recovery

To motivate the filtering mechanism, we recall the baseline 5-round attack of Tiessen [Tie16]. Guessing five subkey bytes across rounds 4 and 5 determines the difference before the S-box in the second to last round ΔSB_0^3 . As there are 2^{72} anchors, only 2^{72} possible d-differences can be reached at this byte. Hence, if we want to filter 2^{40} key candidates, the original polytopic attack needs a 14-difference (which requires a 15-polytope) to end up with 1 key candidate per byte (as you get a $2^{72}/2^{112} = 2^{-40}$ filter).

3.2 Column-wise Sieving

Given the ciphertext polytope $C = \{C_0, C_1, C_2, C_3\}$ and its Round 4 ciphertext difference tuple $\Delta_{out} = (\delta_{out}^1, \delta_{out}^2, \delta_{out}^3)$, we guess a 32-bit diagonal of the Round 4 subkey K_c^4 (so it forms a column after InvShiftRows) and decrypt backwards

through Round 4 **SubBytes** to obtain the differences ΔSB^4 . Because key addition and MixColumns are both linear, this directly yields $\Delta MC^{3'}$.

We filter candidate subkeys by testing whether the recovered $\Delta MC^{3'}$ is reachable under *any* valid assignment of anchor bytes. Because each individual row in [Equation 3](#) depends on only three anchor bytes (24 bits of freedom), an isolated row check provides modest filtering: out of 2^{24} possible 3-differences, S-box differential properties restrict reachability to approximately $1 - e^{-1} \approx 63.2\%$ of the space.

However, enforcing joint geometric consistency across all four rows provides an efficient filter. While each row individually passes $\sim 63.2\%$ of candidates, all four rows **must** be reachable by the **same** Round 1 anchor byte x_0 simultaneously.

When a candidate subkey satisfies Row 0 for an anchor x_0 , this fixes x_0 for Rows 1, 2, and 3, leaving only 2^{16} degrees of freedom per row (y_r, z_r). Because S-box differential propagation across Rows 1, 2, and 3 is algebraically independent, each subsequent row imposes a 2^{-8} filter, yielding a cumulative filtering probability of 2^{-24} across the last three rows. When we apply this filter across the 2^{32} column subkey space, ≈ 256 valid key candidates survive per column.

Cross-Column Anchor Filtering. When we sieve the four columns $c \in \{0, 1, 2, 3\}$ independently, each surviving 32-bit column subkey $K_{col_i}^4$ is compatible with a specific subset of Round 1 anchor bytes $x_0 \in \mathcal{X}$. Because all four columns of the same 128-bit encryption must originate from the identical Round 1 anchor byte x_0 , we avoid testing all $256^4 = 2^{32}$ unconstrained column combinations. Instead, we group surviving column subkeys by their anchor x_0 (see [Figure 3](#)) and construct the 128-bit key candidates via the union of anchor-compatible Cartesian products:

$$\mathcal{K}_{128\text{-bit}} = \bigcup_{x_0 \in \mathcal{X}} \left(\mathcal{K}_{[0..3]}^4(x_0) \times \mathcal{K}_{[4..7]}^4(x_0) \times \mathcal{K}_{[8..11]}^4(x_0) \times \mathcal{K}_{[12..15]}^4(x_0) \right). \quad (4)$$

For any fixed anchor x_0 , the expected number of surviving subkeys per column is ~ 1 . Thus, the expected number of 128-bit key candidates across all 256 anchor buckets is ≈ 256 . We re-encrypt a plaintext to verify these candidates, requiring only ≈ 256 trial encryptions (2^8 4-round AES equivalents), which is computationally negligible compared to the online sieving complexity.

3.3 Complexity and Costs

A naive evaluation of 2^{32} column subkey candidates against all possible anchor combinations would require $4 \times 2^{32} \times 2^{26} = 2^{60}$ checks. However, recomputing reachable differences online is unnecessary. We precompute an offline lookup table T_r that maps every reachable 3-difference $\Delta \in \mathbb{F}_2^{24}$ to the set of compatible Round 1 anchors $x_0 \in \mathcal{X}$.

Because the 3-difference for any individual row relies strictly on a 24-bit parameter space (e.g., x_0, y_0, z_0), populating this table only requires exhaustive

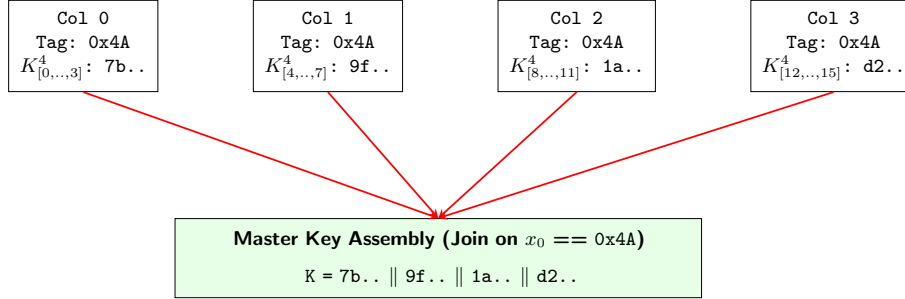


Fig. 3. Cross-Column Anchor Filtering. Instead of evaluating $2^{32} \times 2^{32} \times 2^{32} \times 2^{32}$ combinations, the valid candidate subkeys derived independently per column are grouped by their required root anchor tag (x_0). Only combinations mapping to identical anchors represent realizable states and are assembled into valid 128-bit key candidates.

evaluation of 2^{24} states. This pre-computation executes in 2^{20} 4-round AES equivalents.

During online key recovery, we look up the compatible anchor sets for each row and intersect them. A candidate subkey survives if and only if $T_0[\delta^{(0)}] \cap T_1[\delta^{(1)}] \cap T_2[\delta^{(2)}] \cap T_3[\delta^{(3)}] \neq \emptyset$.

To quantify online sieving complexity, we count the number of table lookups and 4-round AES equivalent operations. Decrypting the 4-polytope across a 32-bit column for 2^{32} key candidates requires 4 column-rounds per candidate, which equals one full 16-byte AES round per candidate ($\approx 2^{30}$ 4-round AES equivalents).

When evaluating a candidate column subkey, we check rows sequentially and abort immediately upon an empty table intersection. Because Row 0 passes $\sim 63.2\%$ of candidates and Row 1 passes $\sim 0.25\%$, the average number of row checks we evaluate per candidate is $1 + 0.632 + 0.003 = 1.635$. Conservatively estimating each row check as 1 AES-round equivalent, the total online sieving complexity across all four columns is therefore:

$$4 \text{ columns} \times 2^{32} \text{ candidates} \times \frac{1 + 1.635}{4} \text{ AES-4 equiv.} \approx 2^{33.39} \text{ 4-round AES. (5)}$$

We filter out the remaining candidates at negligible computational cost: we encrypt one plaintext to immediately discard any remaining false positives. Combining the sieving phase ($2^{33.39}$) and final candidate verification yields an overall time complexity of $\approx 2^{33.39}$ 4-round AES evaluations, securely under 2^{34} .

3.4 Full state sieving with 3 plaintexts

The data requirement can be reduced further to a 3-polytope by exploiting global cross-column dependencies. In the attack described above, it would be impossible to filter key candidates when looking at a single column with a 3-polytope.

However, if we look at *all* anchor bytes at ΔMC^3 across all columns, each state byte only depends on three anchors and the initial difference:

$$\Delta MC^3 = \begin{pmatrix} f(\Delta_{in}, x_0, y_0, z_0) & f(\Delta_{in}, x_0, y_3, z_4) & f(\Delta_{in}, x_0, y_2, z_8) & f(\Delta_{in}, x_0, y_1, z_{12}) \\ f(\Delta_{in}, x_0, y_3, z_5) & f(\Delta_{in}, x_0, y_2, z_9) & f(\Delta_{in}, x_0, y_1, z_{13}) & f(\Delta_{in}, x_0, y_0, z_1) \\ f(\Delta_{in}, x_0, y_2, z_{10}) & f(\Delta_{in}, x_0, y_1, z_{14}) & f(\Delta_{in}, x_0, y_0, z_2) & f(\Delta_{in}, x_0, y_3, z_6) \\ f(\Delta_{in}, x_0, y_1, z_{15}) & f(\Delta_{in}, x_0, y_0, z_3) & f(\Delta_{in}, x_0, y_3, z_7) & f(\Delta_{in}, x_0, y_2, z_{11}) \end{pmatrix}. \quad (6)$$

We denote $\mathbf{A}_{\text{shared}} = (x_0, y_0, y_1, y_2, y_3)$ as the 5 anchor bytes which are shared in each column. While $\mathbf{A}_{\text{shared}}$ provides 40 bits of dependency, the inclusion of the 16 independent Round 3 S-box anchors ($z_0 \dots z_{15}$) yields a total of 168 bits of freedom evaluated across the full state. However, there exist 2^{256} possible 2-differences. This means we still have a theoretical filter of 88 bits. Note that this also explains why simple standard differential cryptanalysis (with a 2-polytope) would be insufficient here.

We create an initial offline lookup table $T[x][r][\Delta MC^3]$, where x is the shared anchor x_0 , r is the row index, and ΔMC^3 the 2-difference. We partition our attack by the shared anchor x , fixing it to a specific value $x_0 \in \mathbb{F}_{2^8}$ and iterating over all 256 possible values.

We populate the table by looping over all possible anchors y_i, z_j (x will be fixed), computing the reachable 2-differences and storing the partial shared anchor x, y_i it can be reached from. We have 2^{16} possible values for y_i, z_j , and 2^{16} possible 2-differences. As this behaves like a random process (see [Subsection A.1](#) for experimental validation), the probability that a random anchor pair produces a specific difference is:

$$P_{\text{valid}} = 1 - e^{-1} \approx 63.2\% \quad (7)$$

If the bucket is non-empty, the conditional expected number of valid y_i traces it contains is:

$$\mu = \frac{\lambda}{1 - e^{-1}} = \frac{1}{1 - e^{-1}} \approx 1.58 \quad (8)$$

In the online phase, for each key candidate we decrypt one round to obtain the 2-difference at ΔMC^3 . We use our lookup table $T[r][\Delta]$ to both filter the key and obtain the set of $\mathbf{Y}_s = (y_0, y_1, y_2, y_3)$ which can be reached by decrypting with this key.

For a 2-difference the filter is relatively weak, pruning around $(1 - e^{-1})^4 \approx 2^{-2.65}$ key candidates per column and leaving approximately $2^{29.35}$ survivors per column. For each surviving key the size of the set $\mathbf{Y}_s = (y_0, y_1, y_2, y_3)$ is $(1.58)^4 \approx 6.23$. This leaves us with $(2^{29.35})^4 = 2^{117.40}$ possible key candidates left, however we do not need to brute-force those.

Because of the dependency established in [Equation 6](#), all four columns must agree on the identical shared $\mathbf{A}_{\text{shared}}$ anchor for the correct master key. We can therefore merge the four columns by joining the four columns table on tag equality:

$$\mathcal{K}_{128\text{-bit}} = \left\{ (K_{\text{col}0}^4, K_{\text{col}1}^4, K_{\text{col}2}^4, K_{\text{col}3}^4) \left| \bigcap_{c=0}^3 Y_s(K_{\text{col}c}^4) \neq \emptyset \right. \right\}. \quad (9)$$

For each column we get $2^{29.35} \cdot 6.23 \approx 2^{32}$ total valid anchors, which are subsequently merged across columns. This 96-bit equality condition across three independent columns filters the combined space, yielding $2^{128} \times 2^{-96} = 2^{32}$ key combinations per fixed slice of x_0 . Across all 256 slices, we aggregate a total candidate pool of 2^{40} keys.

To obtain the final secret key, we therefore only need to test the remaining 2^{40} unique key candidates against one plaintext/ciphertext pair. Note that because of the property of the AES key schedule, we can compute K^0 from K^4 and immediately check whether $P_{0,0} \oplus K_0^0 = x_0$. If it does not match our current slice, we can discard it immediately. This equality check operates as a fast 2^{-8} filter. Applied globally against our 2^{40} candidate combinations, it reduces the candidate space to 2^{32} valid keys overall. The remaining candidates then get filtered by a full 4-round AES encryption. As we process all 2^{40} keys via this filtering, it constitutes the dominant computational cost of this step.

As we have to repeat this attack for all choices of x_0 , the overall costs are dominated by testing a total of 2^{40} key candidates.

4 A 5-round attack on AES

We use a 10-polytope resulting in exactly 9 initial differences at plaintext byte 0. As established in the 4-round sieve, the 16-byte difference state after Round 2 **MixColumns** is uniquely determined by the 40-bit shared anchor core $\mathbf{A}_{\text{shared}} = (x_0, y_0, y_1, y_2, y_3)$.

Entering Round 3, the linear **MixColumns** transforms each single active byte on the diagonal into a fully active output column. By isolating our intersection target exactly to the output of Round 3 **MixColumns** (ΔAK^3), we avoid the unknown addition of subkey K_3 . Because $\Delta SB^4 = \Delta AK^3$, this exact structural difference for a single byte b entering Round 4 **SubBytes** depends on a 72-bit compound parameter:

1. The globally shared anchor $\mathbf{A}_{\text{shared}}$ (40 bits).
2. A local column anchor $\mathbf{Z}_{\text{colc}}$ (32 bits), spanning the four Round 3 anchors exclusively isolated to the column tracing to byte b .

Evaluating the multi-difference requires $40 + 32 = 72$ bits of forward parameters per column. Because our 10-polytope provides 9 differences per state byte, this yields exactly a **72-bit target constraint** (9×8 bits) per targeted byte.

4.1 Backward 40-Bit Extraction

From the given ciphertexts (post-Round 5), computing the 9-difference tuple for exactly one target byte located immediately before Round 4 **SubBytes** requires partially reversing Round 5 and Round 4. Due to **ShiftRows** geometry, this extraction is fully isolated by guessing 40 bits of subkey material:

- 32 bits of K^5 (corresponding to the shifted indices in the ciphertext state).

- 8 bits of K^4 (the specific active byte entering Round 4 MixColumns).

Guessing this 40-bit tuple isolates the target byte, returning the 9-difference tuple ΔSB_0^4 for backward verification.

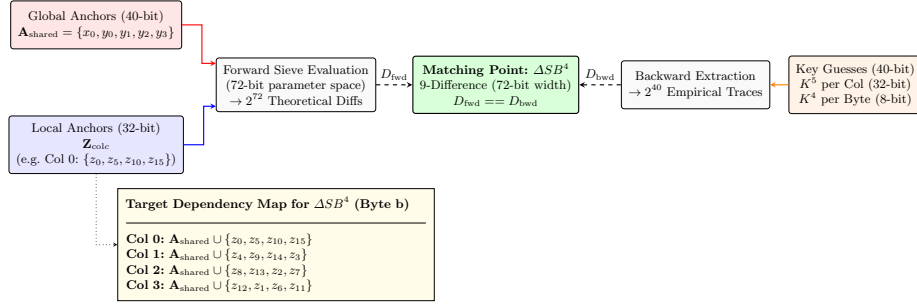


Fig. 4. 5-Round Polytopic Matching Configuration. Evaluating a single column byte at the matching point ΔSB^4 naturally isolates the dependencies. Forward evaluation relies precisely on the 40-bit global root $\mathbf{A}_{\text{shared}}$ (required for all columns) and the 32-bit locally orthogonal column anchor $\mathbf{Z}_{\text{colc}}$. Because a 10-polytope generates exactly 9 active differences per state byte ($9 \times 8 = 72$ bits), the 72-bit forward parameter space intersects against a 72-bit target condition. The backward mapping uniquely extracts this 9-difference by guessing 40 bits of key material, generating exactly 2^{40} reachable backward subset targets.

4.2 Sequential Sieving ($s = 10$)

For each of the 4 target unrolled bytes across the four columns $c \in \{0, 1, 2, 3\}$, we iterate their specific 40-bit key material. We extract the 9-difference tuples and store them in offline tables T_0, T_1, T_2, T_3 . Generating four tables of 2^{40} entries mapping to a 9-difference requires approximately $4 \times 2^{40} \times 9$ bytes $\approx 2^{45.17}$ bytes of storage, or $2^{41.17}$ equivalent AES plaintexts.

With the backward constraint maps generated, we initiate a sequential sieve over the forward 9-difference propagation:

1. **Column 0 Filter:** We establish an outer loop iterating the 2^{40} states of $\mathbf{A}_{\text{shared}}$. For each state, we iterate the inner 32-bit $\mathbf{Z}_{\text{col0}}$ parameter. The resulting 9-difference is queried against the precomputed backward table T_0 (capacity 2^{40}).

This combination tests 2^{72} 9-differences against a 72-bit condition. The expected number of matches S_0 is given by

$$|S_0| = \frac{2^{40}(\mathbf{A}_{\text{shared}}) \times 2^{32}(\mathbf{Z}_{\text{col0}}) \times 2^{40}(\text{table size})}{2^{72}(\text{constraint})} = 2^{40}. \quad (10)$$

We discard all filtered branches, keeping only the $\sim 2^{40}$ valid $\mathbf{A}_{\text{shared}}$ roots forward (note that $\mathbf{Z}_{\text{col0}}$ is independent of the other columns, as it is not a shared anchor with other columns). For each match, we store the mapping $\mathbf{A}_{\text{shared}} \rightarrow K_{\text{col0}}^5$.

2. **Column 1 Filter:** We restrict the subsequent active iteration entirely to the $\approx 2^{40}$ valid $\mathbf{A}_{\text{shared}}$ traces surviving Column 0. Inside each, we iterate Column 1's specific 32-bit anchor $\mathbf{Z}_{\text{col1}}$. Evaluated against T_1 , we expect again that

$$|S_1| = \frac{2^{40} \times 2^{32} \times 2^{40}}{2^{72}} = 2^{40} \quad (11)$$

candidates will survive. For each match, we add the additional key candidates to the stored mapping $\mathbf{A}_{\text{shared}} \rightarrow (K_{\text{col0}}^5, K_{\text{col1}}^5)$.

3. **Column 2 and 3:** Processing is performed analogously, yielding the same amount of expected candidates. In the end we have a complete mapping of key columns which are consistent under their shared anchor $\mathbf{A}_{\text{shared}} \rightarrow (K_{\text{col0}}^5, K_{\text{col1}}^5, K_{\text{col2}}^5, K_{\text{col3}}^5)$.

Note that we choose a 9-difference here, specifically as it keeps the candidates stable at 2^{40} per column. If we decrease the polytope size, it would lead to more candidates, conversely if we increase the polytope size we could directly improve the filtering and reduce the surviving candidates in each column.

When sieving Column 3, the sequence yields strictly $\approx 2^{40}$ globally consistent combinations of forward anchors and key candidates. Across the four columns, each valid entry collected four 32-bit slices of K^5 , yielding a complete 128-bit candidate for K^5 . To finalize key recovery, we enumerate these $\approx 2^{40}$ fully formed K^5 candidates, reversing the AES key-schedule to derive the master key, and test each against a single plaintext-ciphertext pair. This verification phase requires $\approx 2^{40}$ encryptions.

Complexity Summary. The overall time complexity is dominated by computing the forward difference candidates and checking if they are in T_i . Computing the 9-difference forward costs ≈ 2 AES rounds, so in total we get

$$\text{Time Complexity} = 4 \times 2^{72} \times 2/5 = 2^{72.67} \approx \mathcal{O}(2^{73}) \text{ 5-round AES operations.} \quad (12)$$

The memory complexity consists of storing the tables T_i (these are column independent, so we need to store at most one table at a time), which each take $2^{41.17}$ plaintext equivalents. Storing the mapping of shared anchors to key candidates requires exactly 2^{40} plaintext equivalents, so the overall memory complexity is strictly less than 2^{42} .

4.3 Optimized 5-Round Sieve

We can reduce the costs per column by exploiting the local independence of the AES **ShiftRows** and **MixColumns** operations, coupled with a time-memory tradeoff Meet-in-the-Middle. This optimization can reduce the time complexity significantly, while not increasing the stated memory constraints.

Functional Independence. In the standard sieve, the 72-bit target ΔSB^4 per column is evaluated over the 40-bit $\mathbf{A}_{\text{shared}}$ parameter and the 32-bit $\mathbf{Z}_{\text{colc}}$ parameter. However, if we introduce a global outer loop that selectively fixes only the 8-bit reference anchor x_0 , the dependencies propagate differently.

Fixing x_0 causes the 72-bit target difference ΔSB^4 for each column to split into four independent 16-bit branches. We define these localized functions $G_i : \mathbb{F}_2^{16} \rightarrow \mathbb{F}_2^{72}$ as follows (for Column 0, with analogous mappings for other columns):

$$\Delta SB^4 = G_0(y_0, z_0) \oplus G_1(y_3, z_5) \oplus G_2(y_2, z_{10}) \oplus G_3(y_1, z_{15}) \quad (13)$$

Each branch function G_i depends strictly on a 16-bit localized state. This structural decoupling removes the need to evaluate a nested 2^{64} state monolithically; instead, we process the branches independently.

Asymmetric 3-1 Split. For a specific column mapping to our precomputed backward table T_c (which has a capacity of exactly 2^{40} valid 9-differences) with a fixed x_0 , our objective is to find all solutions such that:

$$G_0 \oplus G_1 \oplus G_2 \oplus G_3 = t \in T_c \quad (14)$$

We group the terms to balance a Meet-in-the-Middle condition:

$$G_0 \oplus G_1 \oplus G_2 = T_c \oplus G_3 \quad (15)$$

- **Left Side (U):** Depends on 3 independent 16-bit blocks (48 bits of freedom), generating a 2^{48} state space.
- **Right Side (W):** Depends on the iteration over T_c (2^{40}) and 1 block (2^{16}), generating a 2^{56} state space.

Evaluating this MITM intersection directly would require excessive memory overhead (storing 2^{56} items), therefore we perform this intersection dynamically.

Prefix Time-Memory Tradeoff Sieve (24-bit). Let $\mathcal{P} \in \{0, 1\}^{24}$ represent a 24-bit prefix filter.

1. **Offline Bucketization:**

- *Right Side (T_c):* We reorganize the precomputed backward mapping table T_c by its first 24 bits into 2^{24} buckets. Each bucket houses on average $2^{40}/2^{24} = 2^{16}$ backward difference traces.
- *Left Side ($G_{0,1}$):* For the fixed reference anchor x_0 , we precompute all 2^{32} intermediate sums $G_{0,1}(y_0, z_0, y_3, z_5) = G_0(y_0, z_0) \oplus G_1(y_3, z_5)$ and bucketize them by their 24-bit prefix into a table $H_{0,1}$. Each bucket contains on average $2^{32}/2^{24} = 2^8$ candidate pairs. Populating $H_{0,1}$ requires 2^{32} operations once per slice of x_0 .

2. **Iterative Prefix Sieve:** We iterate sequentially over all 2^{24} prefixes $\mathcal{P}$. For each prefix:

- *Right Filter Generation (W)*: We iterate over the 2^{16} parameter space of $G_3(y_1, z_{15})$. For each value, we query the specific T_c bucket indexed by $\mathcal{P} \oplus \text{prefix}(G_3)$ (size 2^{16}). This deterministic pairing generates exactly $2^{16} \times 2^{16} = 2^{32}$ candidate 72-bit strings matching $\mathcal{P}$ in 2^{32} operations.
- *Left Filter Generation (U)*: We iterate over the 2^{16} parameter space of $G_2(y_2, z_{10})$. For each value, we directly query the bucket in $H_{0,1}$ identically indexed by $\mathcal{P} \oplus \text{prefix}(G_2)$ (size 2^8). This directly yields all $2^{16} \times 2^8 = 2^{24}$ combinations of (G_0, G_1, G_2) that produce prefix $\mathcal{P}$, executing in only 2^{24} operations.
- *Intersection*: We insert the 2^{24} Left-side strings into a compact hash table on their remaining 48 bits and stream the 2^{32} Right-side strings against it. The expected number of surviving 72-bit matches per prefix is $2^{24} \times 2^{32} \times 2^{-48} = 2^8$.

The execution time per prefix is bounded by $\max(2^{24}, 2^{32}) \approx 2^{32}$ operations, dominated by the Right-side generation. Across all 2^{24} prefixes, the total Left-side generation requires only $2^{32} + 2^{24} \times 2^{24} \approx 2^{48}$ operations, making Right-side streaming ($2^{24} \times 2^{32} = 2^{56}$) the primary operational cost per column slice.

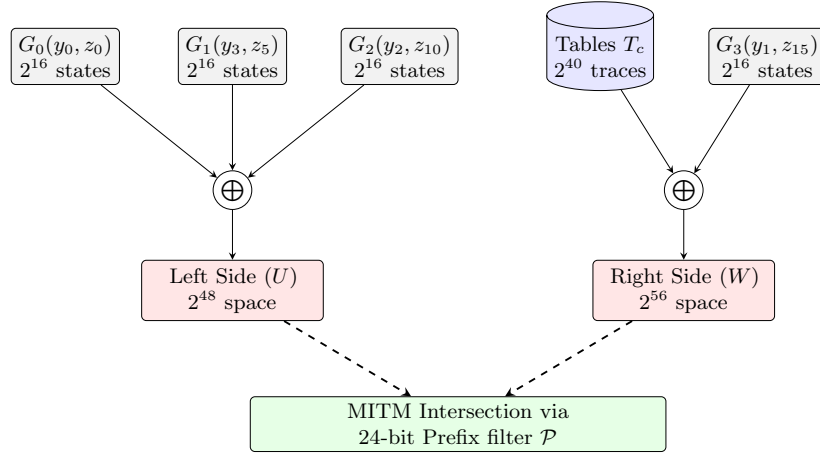


Fig. 5. Asymmetric 3-to-1 dissection sieve for 5-round key recovery. The 72-bit topological condition fractures into independent branches when the reference anchor x_0 is fixed. This prevents monolithic 2^{72} -time evaluations by intersecting a 2^{48} geometric Left space (U) with a 2^{56} geometric Right space (W) dynamically over a 24-bit MITM Prefix filter.

Complexity Summary. The complexity for this approach is given by:

- **Single Column Sieve Time**: Slicing by the 24-bit prefix $\mathcal{P}$ requires evaluating 2^{24} (prefixes) $\times$ 2^{32} (inner iterations) $= 2^{56}$ operations for a fixed root

anchor x_0 . Integrating over all 2^8 values of x_0 yields:

$$T_{\text{col}} = 2^8 \times 2^{56} = 2^{64} \text{ atomic operations per column.}$$

– **Total Atomic Operations across 4 Columns:**

$$T_{\text{total}} = 4 \times 2^{64} = 2^{66} \text{ atomic operations.}$$

– **Conversion to 5-Round AES Encryption Equivalents:** An atomic operation in the inner loop consists of a single table lookup in T_c alongside a 72-bit vector XOR. Evaluating a branch G_i on the fly corresponds to 4 active S-boxes and a partial MixColumns multiplication, which equals 1 column-round of AES. Because a full 5-round AES encryption consists of $5 \times 4 = 20$ column-rounds, one atomic sieve operation corresponds to $\frac{1}{20} \approx 2^{-4.32}$ 5-round AES encryptions. Therefore, the total time complexity is bounded by:

$$T_{\text{AES-5}} = \frac{2^{66} \text{ operations}}{20} \approx 2^{61.68} < 2^{62} \text{ 5-round AES encryptions.}$$

– **Memory Complexity:**

- **Offline Difference Tables ($T_0 \dots T_3$):** Storing all four tables simultaneously requires $4 \times 2^{40} \times 16$ bytes = 64 TB = 2^{46} bytes, which is equivalent to 2^{42} AES plaintexts. When generated and evaluated sequentially, only a single active table of 16 TB (2^{40} plaintexts) is needed at any given time.
- **Active Working RAM:** The 24-bit prefix filter restricts the dynamic Left-side hash table to 2^{24} entries of 8 bytes, requiring strictly 2^{23} plaintext equivalents.
- **Surviving Key Candidate Map:** Storing the 2^{40} surviving candidate mappings $\mathbf{A}_{\text{shared}} \mapsto K_5$ requires $2^{40} \times 16$ bytes = **16 TB** (2^{40} plaintexts).
- **Total Memory Footprint:** Bounded globally by 2^{42} or 2^{40} under sequential table generation.

5 Conclusion

In this work, we present improved polytopic attacks for reduced-round AES by systematically characterizing how anchor bytes govern the propagation and reachability of d-differences across non-linear S-box layers. By showing that geometric consistency across columns drastically restricts the realizable subkey space, we demonstrated that structural reachability can be evaluated with exceptionally low-data requirements.

We applied this framework to 4-round AES to achieve full key recovery using only 3 chosen plaintexts in 2^{40} time, or 4 chosen plaintexts in less than 2^{34} time. Furthermore, we extended the framework to 5-round AES in the independent-key model using a 10-polytope. By using a combination of a Meet-in-the-Middle dissection and a 24-bit prefix filter, we reduced the 5-round time complexity to 2^{62} 5-round AES evaluations with 2^{41} memory.

Future Work. This framework opens several promising avenues for future research:

- **Key Bridging for 6-Round AES:** Incorporating the algebraic relations of the AES key schedule (key bridging) would likely allow extending the polytopic sieve to 6-round AES-128, AES-192, and AES-256 while keeping data requirements exceptionally low.
- **Generalization to Other SPN Ciphers:** The principle of tracking anchor dependencies through non-linear layers applies broadly to substitution-permutation networks (SPNs). Extending polytopic sieving to lightweight block ciphers such as SKINNY, GIFT, and PRESENT [BJK⁺16, BPP⁺17, BKL⁺07] is a natural direction to obtain new low-data attacks.
- **Automated Polytopic Search Tools:** Designing automated constraint-programming (CP) or SAT/MILP frameworks to automatically optimize polytope geometries, anchor partition cuts, and dissection schedules for arbitrary cipher structures.

Acknowledgments. Experiments to validate ideas and implement the attack were conducted with the assistance of LLMs (Gemini 3.1 Pro, Flash 3.7). The optimization to do a 3-1 split for the 5-round attack to reduce complexity has been suggested by the same LLM. The main attack algorithm and strategy/ideas are from the authors.

References

- AES01. Advanced Encryption Standard (AES). National Institute of Standards and Technology, NIST FIPS PUB 197, U.S. Department of Commerce, November 2001.
- BDD⁺10. Charles Bouillaguet, Patrick Derbez, Orr Dunkelman, Nathan Keller, Vincent Rijmen, and Pierre-Alain Fouque. Low data complexity attacks on AES. Cryptology ePrint Archive, Report 2010/633, 2010.
- BDF11. Charles Bouillaguet, Patrick Derbez, and Pierre-Alain Fouque. Automatic search of attacks on round-reduced AES and applications. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 169–187, Santa Barbara, CA, USA, August 14–18, 2011. Springer Berlin Heidelberg, Germany.
- BJK⁺16. Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY family of block ciphers and its low-latency variant MANTIS. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part II*, volume 9815 of *LNCS*, pages 123–153, Santa Barbara, CA, USA, August 14–18, 2016. Springer Berlin Heidelberg, Germany.
- BKL⁺07. Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsøe. PRESENT: An ultra-lightweight block cipher. In Pascal Paillier and Ingrid Verbauwhede, editors, *CHES 2007*, volume 4727 of *LNCS*, pages 450–466, Vienna, Austria, September 10–13, 2007. Springer Berlin Heidelberg, Germany.

- BPP⁺17. Subhadeep Banik, Sumit Kumar Pandey, Thomas Peyrin, Yu Sasaki, Siang Meng Sim, and Yosuke Todo. GIFT: A small present - towards reaching the limit of lightweight encryption. In Wieland Fischer and Naofumi Homma, editors, *CHES 2017*, volume 10529 of *LNCS*, pages 321–345, Taipei, Taiwan, September 25–28, 2017. Springer, Cham, Switzerland.
- BS91. Eli Biham and Adi Shamir. Differential cryptanalysis of DES-like cryptosystems. *Journal of Cryptology*, 4(1):3–72, January 1991.
- CNV13. Anne Canteaut, María Naya-Plasencia, and Bastien Vayssière. Sieve-in-the-middle: Improved MITM attacks. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 222–240, Santa Barbara, CA, USA, August 18–22, 2013. Springer Berlin Heidelberg, Germany.
- DDKS12. Itai Dinur, Orr Dunkelman, Nathan Keller, and Adi Shamir. Efficient dissection of composite problems, with applications to cryptanalysis, knapsacks, and combinatorial search problems. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 719–740, Santa Barbara, CA, USA, August 19–23, 2012. Springer Berlin Heidelberg, Germany.
- Der13. Patrick Derbez. *Meet-in-the-Middle Attacks on AES*. Theses, Ecole Normale Supérieure de Paris - ENS Paris, December 2013.
- DFJ13. Patrick Derbez, Pierre-Alain Fouque, and Jérémy Jean. Improved key recovery attacks on reduced-round AES in the single-key setting. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 371–387, Athens, Greece, May 26–30, 2013. Springer Berlin Heidelberg, Germany.
- DKS10. Orr Dunkelman, Nathan Keller, and Adi Shamir. Improved single-key attacks on 8-round AES-192 and AES-256. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 158–176, Singapore, December 5–9, 2010. Springer Berlin Heidelberg, Germany.
- DS08. Hüseyin Demirci and Ali Aydin Selçuk. A meet-in-the-middle attack on 8-round AES. In Kaisa Nyberg, editor, *FSE 2008*, volume 5086 of *LNCS*, pages 116–126, Lausanne, Switzerland, February 10–13, 2008. Springer Berlin Heidelberg, Germany.
- GM00. Henri Gilbert and Marine Minier. A collision attack on 7 rounds of rijndael. In *AES candidate conference*, volume 230, page 241. New York, 2000.
- GRR16. Lorenzo Grassi, Christian Rechberger, and Sondre Rønjom. Subspace trail cryptanalysis and its applications to AES. *IACR Trans. Symm. Cryptol.*, 2016(2):192–225, 2016.
- GS20. Lorenzo Grassi and Markus Schafneggger. Mixture integral attacks on reduced-round AES with a known/secret S-box. In Karthikeyan Bhargavan, Elisabeth Oswald, and Manoj Prabhakaran, editors, *INDOCRYPT 2020*, volume 12578 of *LNCS*, pages 312–331, Bangalore, India, December 13–16, 2020. Springer, Cham, Switzerland.
- Tie16. Tyge Tiessen. Polytopic cryptanalysis. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part I*, volume 9665 of *LNCS*, pages 214–239, Vienna, Austria, May 8–12, 2016. Springer Berlin Heidelberg, Germany.

A Experimental Verification

A.1 Differential Reachability

To empirically validate the Poisson assumption, we simulated the exact 2-difference reachability across all $2^{16} = 65,536$ anchor pairs (y_i, z_j) for 10 randomly chosen root anchors x_0 and input differences (δ_1, δ_2) . As shown in Table 2, the empirical fraction of reachable 2-differences is $63.135\% \pm 0.07\%$ (closely matching the theoretical $1 - e^{-1} \approx 63.212\%$), and the conditional mean number of valid y_i traces per non-empty bucket is $\mu = 1.5838 \pm 0.002$ (matching the theoretical $\frac{1}{1-e^{-1}} \approx 1.5820$). The bucket occupancy distribution matches the Poisson probability mass function with maximum deviation $< 0.14\%$.

Table 2. Empirical vs. Theoretical Bucket Occupancy for 2^{16} Anchor Pairs (y_i, z_j)

Occupancy k	$k = 0$ (Empty)	$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k \geq 5$
Empirical	36.865%	36.650%	18.484%	6.101%	1.511%	0.389%
Poisson Theory	36.788%	36.788%	18.394%	6.131%	1.533%	0.366%

A.2 Experimental Validation and Performance of 4-round Attack ($s = 4$)

To empirically validate the attack, we implemented and benchmarked an end-to-end 4-round AES key recovery attack using only 4 plaintext/ciphertext pairs. We evaluated the implementation on an AMD EPYC 7B13 workstation running Linux under both single-core (1 thread) and multi-core (64 threads) execution with AVX2 SIMD acceleration. Table 3 presents the empirical measurements for both single-core and 64-core configurations.

Table 3. Empirical single-core and multi-core performance for the 4-round AES-128 polytopic sieving attack ($s = 4$) on an AMD EPYC 7B13 CPU. Times are averaged over 10 key recoveries.

Attack Phase	Single-Core (1 Thread)	Multi-Core (64 Threads)	Parallel Speedup
Offline Precomputation (16×2^{24} entries)	6.095 s	0.455 s	13.4×
Sieving per Column (2^{32} subkeys)	55.86 s	1.75 s	31.9×
Cross-Column Anchor Join	0.08 ms	0.08 ms	1.0×
Total End-to-End Wall Time	229.54 s (≈ 3.8 min)	7.12 s	32.2×

A.3 Experimental Validation and Performance of 4-round Attack ($s = 3$)

When reducing the data requirement from $s = 4$ to $s = 3$, the filtering power per column drops from 2^{-32} to $2^{-2.65}$. Consequently, each column independently produces approximately $2^{29.35}$ valid subkeys and around $2^{31.33}$ canonical 32-bit Round-2 state tags $y = (y_0, y_1, y_2, y_3)$.

Joining four sets of $2^{29.35}$ candidate subkeys presents a memory bottleneck on a standard workstation:

- **Naive Candidate Staging (Out-of-Memory):** Attempting to store all unpruned pairs in memory before cross-column intersection requires allocating large vectors which may exhaust RAM.
- **State-Fixed Slicing (Time-memory trade-off):** Fixing an additional internal Round-2 state byte (such as y_0) simplifies vector storage but increases time complexity by 2^8 .

The Two-Pass Sieve Architecture. To limit concurrent memory usage in our implementation we instead adopt a three staged approach:

1. **Stage 1 (Bitset Sieve):** We allocate two static 512 MB bitsets ($\mathcal{B}_0, \mathcal{B}_1$) covering the 2^{32} tag space. Column 0 marks active tags into $\mathcal{B}_0$ with atomic operations. Subsequent columns $c \in \{1, 2, 3\}$ mark candidate tags into $\mathcal{B}_1$ only if already present in $\mathcal{B}_0$, followed by a bitwise intersection $\mathcal{B}_0 \leftarrow \mathcal{B}_0 \& \mathcal{B}_1$. This progressively filters the tag space.
2. **Stage 2 (Targeted Bucket Extraction):** Using the surviving bitset $\mathcal{B}_0$, each column is re-evaluated to materialize only the subkeys whose tags survived all 4 columns, partitioning them into 256 buckets by the leading byte $y_0 = (y \gg 24)$ and sorting each bucket in parallel.
3. **Stage 3 (Bucket Merge-Join & AES-NI Verification):** A parallel 4-way merge join operates over the 256 independent buckets. Matching 4-column candidates undergo a consistency check ($K_0^0 \oplus P_{0,0} \stackrel{?}{=} x_0$) and are verified via hardware AES-NI trial encryption.

Experimental Timing Results The implementation was benchmarked on an AMD EPYC 7B13 workstation (64 vCPUs). Table 4 summarizes the wall-clock execution time across each attack phase for a single x_0 slice.

Complexity. A single x_0 slice executes in 148.04 seconds (≈ 2.47 minutes) with a peak memory footprint of 54.24 GB (safely below 64 GB host RAM). Searching all 256 possible values of x_0 yields an expected full 128-bit master key recovery time of $128 \times 148 \text{ s} \approx 5.26$ hours on a single multi-core workstation.

Table 4. Execution time breakdown for the ($s = 3$) polytopic attack on 4-round AES-128 (64 threads, single x_0 slice).

Phase	Description	Time
Phase 1	Precomputing Reachability Tables	1.30 s
Phase 2	Progressive 4-Column Bitset Sieve	66.65 s
Phase 3	Targeted Bucket Extraction & Sorting	76.90 s
Phase 4	256-Bucket Merge Join & Trial Verification	0.17 s
Total	Single x_0 Slice Key Recovery	148.04 s

B Data-Time-Memory Tradeoff: BFS vs DFS Evaluation

The requirement of exactly $s = 10$ chosen plaintexts (establishing a 10-polytope with $d = 9$ differences) is not an arbitrary boundary; it mathematically defines the algorithm’s steady-state. By examining the algorithm under both a Breadth-First Search (Sequential) and a Depth-First Search (On-the-fly) strategy, we rigorously prove why $s = 10$ is the theoretical lower bound for data complexity in the 5-round independent-key sieve before encountering excessive dimensionality growth.

At any given column stage, the forward parameter space comprises 2^{40} states for $\mathbf{A}_{\text{shared}}$ and 2^{32} states for $\mathbf{Z}_{\text{colc}}$, generating 2^{72} localized states. These are checked against a backward table of size 2^{40} . For a polytope with d differences, the probability is $p = 2^{40} \times (2^{-8})^d = 2^{40-8d}$.

We compare two traversal strategies to evaluate the columns:

B.1 Strategy 1: Sequential Breadth-First Search (BFS)

In the BFS strategy (as defined in [Subsection 4.3](#)), the algorithm evaluates a column, stores all surviving candidates in an intermediate list S_i , and expands them in the next column by iterating the subsequent 2^{32} states of $\mathbf{Z}_{\text{colc}}$. The number of evaluations at step i is $N_i = S_{i-1} \times 2^{32}$, and the subsequent list size becomes $S_i = N_i \times p$.

- **At $s = 10$ ($d = 9$):** The filter $p = 2^{-32}$. The geometric expansion rate (2^{32}) exactly offsets the filter probability (2^{-32}). Thus, S_i remains pinned at 2^{40} elements across all columns. This is the constant steady-state, yielding $\approx \mathcal{O}(2^{74})$ time and trivial memory overhead ($\approx \mathcal{O}(2^{40})$ storage).
- **At $s < 10$:** The filter weakens. For $s = 9$ ($d = 8$, $p = 2^{-24}$), the list grows by a factor of 2^8 per column. By Column 3, storing the intermediate parameter combinations requires 2^{64} elements. Upon resolving the final 4th column to extract the remaining key variables, the list geometrically expands to 2^{72} elements, violating practical memory bounds. For $s = 7$, this cascade extends to 2^{144} time and 2^{136} terminal memory elements.

B.2 Strategy 2: On-the-fly Depth-First Search (DFS)

To circumvent the memory scaling of BFS, we can evaluate the same search as DFS. In DFS, we fix a single 40-bit $\mathbf{A}_{\text{shared}}$ root and evaluate all four columns independently on-the-fly, retaining no large intermediate sets. For a fixed $\mathbf{A}_{\text{shared}}$, the expected number of valid, surviving backward keys per column is $m = 2^{32} \times 2^{40} \times (2^{-8})^d = 2^{72-8d}$. Because the columns are structurally independent parameters, forming a valid 128-bit key candidate for terminal verification requires assembling every combination of these column branches. Thus, for each $\mathbf{A}_{\text{shared}}$, we must perform m^4 trial encryptions. Total verification time across the global loop is $2^{40} \times m^4$.

- **At $s = 10$ ($d = 9$):** The number of surviving keys per column is $m = 2^{72-72} = 1$. The combinations per $\mathbf{A}_{\text{shared}}$ collapse to $1^4 = 1$. Total verification operations are $2^{40} \times 1 = 2^{40}$, rendering the verification phase hidden beneath the dominant $\mathcal{O}(2^{62})$ (5-round AES equivalent) sieving time.
- **At $s < 10$:** The memory overhead is mitigated, but the time complexity shifts significantly to the terminal verification phase. For $s = 9$ ($m = 2^8$), total verification tests rise to $2^{40} \times (2^8)^4 = 2^{72}$ trial encryptions. For $s = 7$ ($m = 2^{24}$), verifying the false positive combinations necessitates 2^{136} trial encryptions, exceeding the $\mathcal{O}(2^{128})$ exhaustive bound.

B.3 Parametric Complexity Scaling Summary

Table 5 summarizes the scaling across both strategies. Note that while the optimized sieve used in Subsection 4.3 can be used independently of the choice s , for $s < 10$ the computational bottleneck shifts entirely from sieving to the final key verification phase.

Table 5. Complexity Scaling across BFS and DFS approaches.

Data (s)	Diffs (d)	Constraint (2^{-8d})	BFS Time	BFS Mem.	DFS Verification Time	DFS Mem.
10	9	2^{-72}	$\mathcal{O}(2^{74})$	$\mathcal{O}(2^{40})$	$\mathcal{O}(2^{40})$	$\mathcal{O}(2^{40})$
9	8	2^{-64}	$\mathcal{O}(2^{96})$	$\mathcal{O}(2^{72})$	$\mathcal{O}(2^{72})$	$\mathcal{O}(2^{40})$
8	7	2^{-56}	$\mathcal{O}(2^{120})$	$\mathcal{O}(2^{104})$	$\mathcal{O}(2^{104})$	$\mathcal{O}(2^{40})$
7	6	2^{-48}	$\mathcal{O}(2^{144})$	$\mathcal{O}(2^{136})$	$\mathcal{O}(2^{136})$	$\mathcal{O}(2^{40})$