Quotienting Redundant Tables: Practical Master-Key Recovery from WBSLT White-Box AES

Jiseok Bang Seungkwang Lee

Department of Cybersecurity, Dankook University
Yongin, Republic of Korea

jsbang@dankook.ac.kr sk.cryptographic@dankook.ac.kr

Abstract

Enlarging lookup-table addresses does not hide smaller selectors when the complete public table factors through a core with distinct rows and distinct columns. We demonstrate this condition in WBSLT, a white-box AES design presented at NDSS 2026 that reports resistance to several structural attacks. In the released general generator, the joint output of each group of four first-round TypeII tables has an exact 16×16 quotient of its 64×64 address space. Mapping its row and column classes to observed addresses recovers the original nibble partition of each AES input byte. Public tables implementing encoded XOR supply compatible output coordinates. Under the generator's strictly linear input decoder, two 16-point restrictions of each keyed AES S-box then identify its embedded key byte; independent exhaustive checks confirm this property for all 256 byte keys. Using public tables and traces of chosen encoded inputs, we recover every key in 200 general instances that passed 200,000 reference AES block tests, after a median of 146 and at most 221 inputs. Separate 200-instance cohorts of the released optimized C source and its one-line functional repair also yield every key; only the repair passes the AES tests. An adjacent-round adaptation recovers 20/20 exact keys on a paired cohort without reconstructing the nibble partition. On those instances, our first-round route uses fewer live trace queries and less public table data. The exact quotient provides a concrete test for selector partitions that survive address enlargement.

Keywords: white-box cryptography, AES, key extraction, lookup tables, external encoding.

1 Introduction

A white-box implementation must keep a secret key hidden even when the adversary can inspect its code, lookup tables, and intermediate computations. This is a demanding but deliberate model: an attacker who controls local execution can often collect internal addresses as well as external input–output pairs. The original table-based AES constructions made this setting concrete [1, 2]; later structural attacks and attacks on computation traces showed why the security of internal encodings is difficult to assess [3–5]. We focus on the narrow, testable goal of extracting the exact AES-128 master key. That goal differs from proving that a white-box program can be copied, compressed, or used to decrypt without exposing its key [6, 7].

WBSLT is a recent framework for substitution–linear transformation ciphers, with an AES implementation and public source code [8, 9]. It expands table addresses, mixes encoded components, and encodes external inputs. Its published analysis argues that the transformed table boundaries prevent isolation of key-dependent functions, reports 40,000 unsuccessful attempts across tested BGE, MGH, LRDRP, and BS adaptations, and analyzes the cost of recovering the

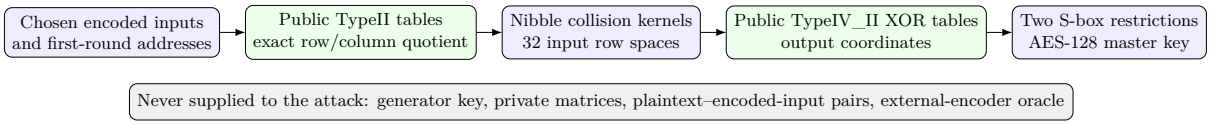

Figure 1: Attack boundary and data flow. Blue boxes show stages that use execution traces; green boxes use only public tables. The hidden material below the flow is not an attack input.

complete external encoding through a particular route [8]. We ask whether substantially less information suffices for exact key extraction: can public table structure and first-round addresses reveal the input partitions and output coordinates needed to identify the key? We answer this question for the released general generator and separately evaluate the optimized C generator.

The released WBSLT-general generator has such a route. Although a first-round TypeII table accepts a 12-bit address, each of its two six-bit selectors has only 16 distinct behaviors when all four output chunks are considered together. Comparing complete public rows and columns reveals these selector classes exactly. Observing their labels at first-round addresses for chosen *encoded* inputs then reveals the 32 four-dimensional row spaces of the hidden linear input decoder. Public tables that combine outputs by disguised XOR provide coordinates for the encoded TypeII outputs. The resulting byte functions are injective affine images of keyed AES S-box values; an exhaustive finite test shows that their two 16-point nibble restrictions determine each key byte. More generally, whenever a public table factors through a smaller core with distinct rows and distinct columns, complete table equality exposes the selector partition regardless of the enlarged address space.

Our contributions are a criterion for recovering hidden selector partitions from complete public table equality, a first-round attack that uses these partitions to identify the embedded AES key, and validation on the released general and optimized C generators (Figure 1). Independent exhaustive checks establish uniqueness of each key byte under the stated decoder assumptions. Experiments distinguish the released optimized source from its functional repair and compare the first-round route with an adjacent-round alternative on the same targets.

Our analysis builds on established methods for recovering affine encodings and partial input spaces [5, 10, 11]. Derbez *et al.* recover equivalent affine layers from a forward S-box oracle and use adjacent rounds to identify a key in a different implementation. We adapt adjacent-round matching to WBSLT using whole-byte output representatives; this route recovers exact keys on 20 instances. The first-round route uses different information: its row and column classes reveal the original nibble partition, which resolves the embedded key label under the stated decoder condition without a second round. Section 4.2 isolates what the partition contributes.

2 Target and Threat Model

AES and the implementation AES-128 has a 128-bit master key and uses the byte permutation $\mathsf{S} : \mathbb{F}_2^8 \to \mathbb{F}_2^8$ in its first round [12]. We target the released `E2-AES-Performance/WBSLT-general` code at commit `3c9fdfd3` [9]. The full commit digest is recorded in the artifact provenance record. Our claim applies to that source and the instances it generates. Section 5.1 validates their documented encode–encrypt–decode path against reference AES.

For each first-round byte slot j, the general implementation creates four TypeII chunks $T_{j,t}$, $0 \le t < 4$. Each chunk has a 12-bit address, split into two six-bit selectors (a, b), and a 32-bit output. The four outputs form parts of a 128-bit contribution to the first round. Public TypeIV_II tables in the first stage combine encoded nibbles by disguised XOR. The external input decoder is an invertible *linear* map on $\mathbb{F}_2^{128}$; no affine offset is added at this boundary. We verify these facts in the pinned generator rather than assuming they hold for every implementation of the framework.

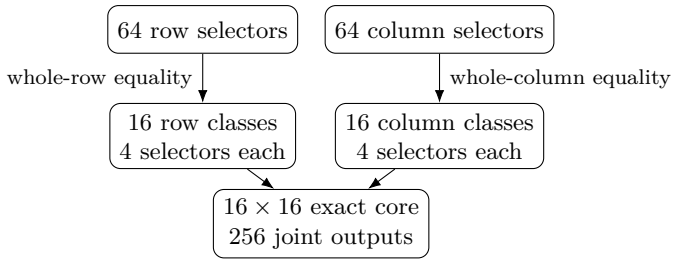

Figure 2: Quotient of one byte slot's 64×64 public address space. Equality compares the *joint* outputs of all four TypeII chunks. The core exactly reconstructs every original chunk entry.

Adversary and success condition The adversary can read the program's lookup tables, choose 128-bit values at its interface for *encoded inputs*, run the same generated instance repeatedly, and record its first-round TypeII addresses. This access falls within the local white-box model discussed by WBSLT [8]. It is stronger than remote ciphertext-only access, but does not provide the generator's private files: the attack receives no true key, private matrices or permutations, correspondence between plaintexts and encoded inputs, or oracle that converts arbitrary plaintexts to encoded inputs. The generator key appears only in the evaluation harness, where it is compared with the recovered key after the attack. Success means exact recovery of all 128 bits of the master key, in AES byte order, for the same generated instance.

Our instrumented collector records the 16 selector words before the first `ShiftRows` and then returns; the fixed permutation gives their TypeII indices. Section 5.1 validates these observations against the unmodified encryption function on the same tables and inputs.

Assumption boundary The proof relies on four conditions: (i) each first-round six-bit selector has 16 effective values; (ii) the joint output of the four chunks is injective in the resulting byte; (iii) the external decoder is invertible and strictly linear, and the two selectors correspond to the original AES byte's upper and lower nibbles; and (iv) each public table in the first stage implements a disguised four-bit XOR. We check the table identities, quotient sizes, and output ranks from public data. We verify condition (iii) in the pinned generator's source; collisions alone cannot distinguish a linear decoder from one with an unknown affine offset. If a checked condition fails, the procedure rejects the instance rather than reporting a candidate as a key. Changes to an encoder or table network can invalidate this route and would require a new security analysis.

3 Recovering the Input Nibble Partition

3.1 Exact Quotient of the First-Round Tables

For one byte slot, concatenate the four public chunk values:

$$T_j(a,b) = (T_{j,0}(a,b), \ldots, T_{j,3}(a,b)) \in \mathbb{F}_2^{128}. \tag{1}$$

Define $a \sim_r a'$ when $T_j(a,b) = T_j(a',b)$ for every b, and define $b \sim_c b'$ symmetrically. We compute these relations by comparing all 64 public row vectors and 64 column vectors for exact equality. Let r_j and c_j map each six-bit selector to its class, and choose one representative of each class. The core table C_j contains the representative entries and satisfies

$$T_j(a,b) = C_j(r_j(a), c_j(b)) \quad \text{for all } a, b. \tag{2}$$

Why the factorization is exact Replacing a by any representative of its row class leaves every output unchanged; the same holds when b is replaced by a representative of its column class. These replacements prove Eq. (2) without statistical assumptions. Any factorization through independent row and column selectors must also assign different labels to rows or columns with different behavior. The quotient is therefore minimal *within that factorization form*; this is not a general lower bound on compression.

When equality reveals the hidden selectors The quotient argument applies more generally than the four-to-one map in WBSLT. Let $\rho : \mathcal{A} \twoheadrightarrow \mathcal{U}$ and $\kappa : \mathcal{B} \twoheadrightarrow \mathcal{V}$ be onto maps, and suppose

$$T(a, b) = G(\rho(a), \kappa(b)). \tag{3}$$

Assume that G separates its rows and columns: for every $u \neq u'$ there is a v with $G(u, v) \neq G(u', v)$, and for every $v \neq v'$ there is a u with $G(u, v) \neq G(u, v')$. Then two complete rows of T are equal exactly when their ρ values are equal; the analogous statement holds for κ and columns. Thus the public quotient recovers both hidden selector partitions, up to independent bijective relabeling.

To prove the row statement, equal ρ values give identical rows by Eq. (3). Conversely, distinct ρ values have a distinguishing v by row separation, and surjectivity of κ supplies a public column with that value. The column argument is symmetric. The separation conditions are also necessary for recovering these particular partitions: indistinguishable core rows or columns merge their respective fibers. Joint injectivity of G is a sufficient, stronger condition, as in the WBSLT specialization below.

The fibers need not have equal sizes. Independent permutations of the public row and column addresses only permute the recovered classes, and an injective relabeling of the joint output preserves equality exactly. This proposition identifies the table condition that exposes selectors; subsequent kernel recovery requires affine selector values in the chosen inputs, and exact key recovery additionally requires the cipher-specific relation test.

WBSLT's partial adders reduce each six-bit selector to four bits [8]. In the pinned generator, this reduction applies a secret permutation followed by the surjection

$$\psi(z) = ((z \gg 2)\ \&\ 12)\ \mid\ ((z\ \&\ 3) \oplus ((z \gg 2)\ \&\ 3)). \tag{4}$$

Each of its 16 values has four preimages. In the pinned generator, the two ψ outputs form a byte that is XORed with the first-round key byte. The generator then applies the AES S-box, the byte's MixColumns contribution, a 128-bit mixing matrix, and independent encodings to the output nibbles. With the other selector fixed, different ψ values give different S-box inputs. The MixColumns contribution is injective for one byte, while the later mixing and encodings are invertible; these maps preserve distinguishability across the *joint* output of the four chunks. Hence each selector has exactly 16 classes of four addresses, and the 16×16 core has 256 distinct outputs. An individual 32-bit chunk need not distinguish all 16 values; our factorization compares the joint output and then verifies every original chunk entry.

Figure 2 illustrates the quotient. Besides reducing the table size, it provides class labels for addresses observed during execution. The labels are arbitrary permutations of hidden nibble values, but their equality is meaningful and sufficient for the next step.

3.2 Linear Input Subspaces from Address Collisions

Let $x \in \mathbb{F}_2^{128}$ be a chosen encoded input. For each of the 32 nibble selectors in the first round, the underlying value is $A_i x \in \mathbb{F}_2^4$, where A_i has rank four. The observed quotient label is

$$f_i(x) = \pi_i(A_i x) \tag{5}$$

for an unknown permutation π_i of 16 values. Equation (5) follows from the pinned generator and the exact quotient; it does not rely on interpreting raw 12-bit addresses as hidden bytes. If $f_i(x) = f_i(x')$, injectivity of π_i implies $A_i(x \oplus x') = 0$. Thus the difference between two inputs with the same label lies in $\ker A_i$, a 124-dimensional space.

For each label, retain one input and XOR it with the other inputs bearing that label. The span K_i of these differences is contained in $\ker A_i$. If $\dim K_i = 124$, the two spaces are equal. The orthogonal complement $K_i^{\perp}$ is the four-dimensional row space of A_i. Any basis B_i of this space suffices: $B_i = M_i A_i$ for some $M_i \in \mathrm{GL}(4, 2)$. A recovered coordinate $u = B_i x$ is therefore a linear, invertible relabeling of the hidden nibble. In particular, $u = 0$ represents the *semantic* nibble zero even if its quotient-class label is not zero. Pairing the two nibble spaces in a byte gives an eight-dimensional input space for that byte.

Sample completion With q independent uniform encoded inputs, each nibble yields at least $q - 16$ differences between inputs with the same label. Conditional on the observed labels, these differences are independent uniform elements of the true kernel. A union bound over its nonzero linear functionals and then over 32 nibbles gives

$$\Pr[\text{some kernel rank deficient}] \leq 32 \cdot 2^{140-q}. \tag{6}$$

To justify the bound, condition on the observed labels of one nibble. Within each label group, the sampled inputs are independent uniform points in one affine coset of the 124-dimensional kernel. XOR the first point with each later point. Conditional on the first point, the resulting vectors are independent uniform kernel elements, and their joint distribution does not depend on that point. If $c \leq 16$ labels occurred, all groups together give $m = q - c \geq q - 16$ such vectors. If they fail to span the kernel, some nonzero linear functional annihilates all m vectors. A union bound over the $2^{124} - 1$ possible functionals gives probability less than $2^{124-m} \leq 2^{140-q}$. A second union bound over 32 nibbles gives Eq. (6), without assuming independence between nibbles. The implementation also needs the observed label for each of the 16 coordinates of every nibble. A separate coupon bound is

$$\Pr[\text{some label unseen}] \leq 512(15/16)^q. \tag{7}$$

A fixed label is absent with probability $(15/16)^q$; the factor 512 comes from the union bound over 32×16 nibble-label pairs. The two bounds describe separate requirements, and both must be considered. They approach zero as q grows, so continued independent uniform sampling reaches complete rank and label coverage almost surely for an instance satisfying the verified table conditions. The attacker checks both conditions on the actual trace and collects more inputs from the *same* instance until they hold. The two nibble spaces are used separately, so the full input decoder need not be guessed.

4 Identifying the AES Key

4.1 Output Coordinates and the Observable Byte Function

Each public TypeIV_II table used in the first stage has 16×16 entries of the form

$$F(s, t) = \gamma(\alpha(s) \oplus \beta(t)), \tag{8}$$

where α, β, γ are unknown nibble permutations. We recover coordinates without guessing them. Fix a row s_0 and column t_0, and let $p(s) = F(s, t_0)$ and $q(t) = F(s_0, t)$. Both are permutations. Define a binary operation on public output labels by

$$z \star w = F(p^{-1}(z), q^{-1}(w)). \tag{9}$$

Writing $e = F(s_0, t_0)$, substitution into Eq. (8) shows that $(\operatorname{im} F, \star)$ is a four-dimensional elementary abelian group with identity e. Choose any group basis and call its coordinate map ϕ. Then

$$F(s,t) = \phi^{-1}(\ell(s) \oplus \rho(t)), \tag{10}$$

where $\ell(s) = \phi(p(s))$ and $\rho(t) = \phi(q(t))$. The implementation verifies Eq. (10) for all 256 entries of every public table it uses. More explicitly, if $a_0 = \alpha(s_0)$ and $b_0 = \beta(t_0)$, then $\phi(\gamma(z)) = U(z \oplus a_0 \oplus b_0)$ for some invertible linear U. Thus an invertible affine map relates the recovered coordinates to the producer's decoded nibble coordinates. We retain the constant introduced by the chosen row and column instead of assuming that public label zero represents decoded zero.

The recovered input basis determines which public class represents each of its 16 coordinate values. For selector i, record $\tau_i(B_i x) = f_i(x)$ for every sampled input x. Because $B_i = M_i A_i$ and $f_i = \pi_i \circ A_i$, this is a well-defined bijection from the recovered coordinates to the public classes; the coverage check supplies all 16 entries. Let $\tau_{j,r}$ and $\tau_{j,c}$ be these maps for the row and column selectors of byte slot j. For every $(u, v) \in \mathbb{F}_2^4 \times \mathbb{F}_2^4$, we can now read

$$\widetilde{T}_j(u,v) = C_j(\tau_{j,r}(u), \tau_{j,c}(v)) \tag{11}$$

from the public core, without further execution queries. The recovered factors of the first-stage TypeIV_II tables convert each of its 32 encoded output nibbles into compatible XOR coordinates. Concatenating them gives a 128-bit function of (u, v). In the pinned generator, that function has the form

$$V_j(u,v) = H_j \mathsf{S}(k_j \oplus L_{j,0} u \oplus L_{j,1} v) \oplus b_j. \tag{12}$$

Here $u, v \in \mathbb{F}_2^4$ are the recovered *linear* input coordinates. The maps $L_{j,0}$ and $L_{j,1}$ are unknown invertible changes of basis for the two nibbles, embedded into complementary halves of a byte; k_j is the first-round key byte; and $H_j : \mathbb{F}_2^8 \to \mathbb{F}_2^{128}$ is an injective linear map. Each byte's contribution to MixColumns has rank eight, and both the generator's 128-bit mixing and the recovered changes to nibble coordinates are invertible. Hence $\operatorname{rank} H_j = 8$ when all four chunks are concatenated. The constant b_j arises from arbitrary choices of public XOR reference points and disappears when we center the outputs.

We check that the differences among the 256 reconstructed outputs span an eight-dimensional space, then project them to eight coordinates by Gaussian elimination. Because this projection is invertible *on that span*, it preserves every affine relation among the observed points. An individual chunk need not have rank eight; its output alone would not justify this step.

4.2 What the Nibble Partition Adds

The public TypeII tables, recovered XOR coordinates, and observed addresses allow a comparison with generic recovery of affine encodings [5]. First, decode the distinct outputs of each byte table using the public XOR factors in the first stage, and project their eight-dimensional affine span to one byte. For any chosen encoded input, the observed TypeII address selects one projected output per byte. Concatenating the 16 outputs yields a forward 128-bit permutation of the form $F = B \circ (\mathsf{S}, \ldots, \mathsf{S}) \circ A$, with unknown affine A, B: the pinned input decoder is invertible, and each output projection is bijective on its byte span. This oracle can be computed without recovering the nibble kernels or first computing the row and column classes. It has the form considered by Derbez *et al.*; the raw 64×64 table itself need not be a permutation. Their generic objective is an *equivalent* pair of affine layers, which need not be the generator's original pair. Their separate attack on a different white-box design also recovers its key; our distinction concerns the WBSLT interface studied here.

An equivalent byte oracle alone need not determine this generator's actual first-round key byte. Write the AES S-box as $\mathsf{S}(z) = J(z^{-1}) \oplus d$, where J is invertible and linear over $\mathbb{F}_2$, with

$0^{-1} = 0$. For every nonzero $a \in \mathrm{GF}(2^8)$, the affine permutation

$$P_a(y) = J(a^{-1}J^{-1}(y \oplus d)) \oplus d \tag{13}$$

satisfies $\mathsf{S}(az) = P_a(\mathsf{S}(z))$, including at zero. Consequently, an oracle $H\mathsf{S}(Mx \oplus k) \oplus c$ that does not reveal the nibble partition has the same input–output behavior if M, k are replaced by aM, ak and the unknown affine output coordinates are adjusted by P_a^{-1}. The decoder remains linear. For nonzero k, these representations include all 255 nonzero byte labels as a varies. This ambiguity concerns only the normalized first-round oracle considered in isolation; it is not a bound on attacks that also use other rounds or retain the underlying WBSLT tables and addresses. Our exact-key attack uses the partition information in those additional observations.

The public quotient, *combined with addresses observed for chosen inputs*, provides information beyond that oracle: collisions identify the two original nibble kernels separately. Because the pinned decoder preserves zero, the recovered coordinates may change independently by $\mathrm{GL}(4, 2)$ within each nibble, but a $\mathrm{GL}(8, 2)$ change that mixes the two observed spaces is inadmissible. To see why this removes the scaling ambiguity, work in the AES field $\mathbb{F}_2[x]/(x^8+x^4+x^3+x+1)$ and write $U = \mathrm{span}_{\mathbb{F}_2}\{1, x, x^2, x^3\}$ and $V = x^4U$. If $aU = U$ for nonzero a, then $a \in U$; a nonconstant a of degree $d \leq 3$ sends $x^{4-d} \in U$ to a polynomial of degree four, so $a = 1$. If a exchanges U and V, then $aU = x^4U$ gives $a = x^4b$ with $bU = U$, hence $a = x^4$; but aV contains $x^8 = x^4 + x^3 + x + 1 \notin U$. Thus only the identity scaling preserves the observed pair, even allowing exchange. This removes the scaling ambiguity; the finite AES check below establishes full key uniqueness under all admissible changes of nibble basis.

The distinction is also constructive. Our separate analysis based on F alone obtains an affine-equivalent byte representation without the row/column quotient. For a byte map $f(u) = B\mathsf{S}(Mu \oplus k) \oplus b$ with invertible linear B, M, let $t = M^{-1}k$. For every nonzero input difference d, the differential distribution table of f has a unique output difference of count four, denoted $T(d)$:

$$T(d) = BJ((Md)^{-1}) = f(t \oplus d) \oplus f(t). \tag{14}$$

Thus t is identifiable in the input basis obtained from F alone by checking Eq. (14) for all $d \neq 0$. A wrong point w with $r = M(w \oplus t) \neq 0$ can satisfy the equation only for $Md = r$ or for the two roots of $(Md/r)^2+(Md/r)+1 = 0$. Set $T(0) = 0$ and define $\sigma_v(x) = T^{-1}(T(x)\oplus T(x\oplus v))\oplus x$ for $x \notin \{0, v\}$, with $\sigma_v(0) = 0$ and $\sigma_v(v) = v$. In hidden field coordinates, $\sigma_v(x) = x^2/v$; choosing coordinate 1 as identity makes $\sigma_1 \circ \sigma_v^{-1}$ multiplication by v. A root of the AES field polynomial yields an equivalent affine–S-box representation, but its key label need not be the original one.

Starting from that representative, we enumerate the 2,040 input maps $E_{a,i}(z) = az^{2^i}$, $a \in \mathrm{GF}(2^8)^*$ and $0 \leq i < 8$, along with the corresponding affine compensations at the S-box output. For each transformed map, we compare its row spaces for the low and high nibbles with those recovered from public classes and address collisions. This comparison fixes the original key label within the enumerated family. The finite test in Section 4.3 establishes key uniqueness under all admissible nibble bases, independently of whether this family exhausts all affine self-equivalences. Section 5.3 evaluates an alternative that uses relations between rounds.

4.3 Two Nibble Relations Identify the Key

Restrict Eq. (12) first to $v = 0$ and then to $u = 0$. For a list of 16 points $g(0), \ldots, g(15)$, form a centered matrix with columns $g(i) \oplus g(0)$, $1 \leq i < 16$. Its kernel is the space of linear relations among these 15 differences. An injective affine map of the outputs preserves this kernel exactly. For each candidate byte h, each nibble position $s \in \{0, 4\}$, and every $M \in \mathrm{GL}(4, 2)$, we compare the observed kernel with that of

$$g_{h,s,M}(u) = \mathsf{S}(h \oplus (Mu \ll s)). \tag{15}$$

Table 1: Exhaustive finite checks of the AES S-box. Two programs use different enumeration methods and signature representations; both give these results. This check is distinct from the evaluation of generated instances.

Quantity	Verified value
Invertible four-bit maps	20,160
Candidate parameterizations	10,321,920
True byte keys tested	256
Two-restriction singleton intersections	256
Missing true byte keys	0
Maximum candidates from one restriction	2

We require both centered matrices to have rank eight and the same relation space. When these conditions hold, an invertible linear map relates their output difference spans, so the test accounts exactly for the unknown output basis in Eq. (12). The input bases account for the unknown $L_{j,0}$ and $L_{j,1}$, and testing both positions accommodates the unknown assignment to the upper and lower nibbles. The intersection of the two sets of candidate keys is the recovered k_j.

Finite AES property For the specified AES S-box, the two nibble restrictions together uniquely identify every true byte key under arbitrary invertible four-bit input bases and injective affine output maps. We establish this as a finite, computer-assisted result, rather than assuming it holds for all S-boxes. The verifier enumerates all $|\,\mathrm{GL}(4,2)| = 20{,}160$ bases, 256 candidate keys, and two nibble positions: 10,321,920 candidate parameterizations. For each of the 256 true keys, the candidate sets from the two canonical restrictions intersect only at the true key; a set from either nibble alone can contain two keys. An arbitrary input basis only permutes the 16 indices of a restriction, while the candidate search already covers all bases; injective affine output maps preserve centered relations. An independent checker enumerates invertible matrices in a different way and compares row spaces rather than relation kernels; it also finds 256 singleton intersections.

Procedure and rejection conditions For each generated instance, the attack (1) computes and verifies the 16-class quotient of the joint TypeII chunks; (2) maps observed addresses in the first round to class labels; (3) collects differences between inputs with the same label until the differences for every nibble span a space of dimension 124 and every coordinate has an observed label; (4) factors and verifies the public XOR tables in the first stage; (5) reconstructs each 256-point byte function and checks for rank eight; and (6) applies the relation test to both nibble restrictions, requiring one key byte per slot. It then reverses the known ordering of bytes in the first round to obtain one AES-128 master-key candidate. The observable stopping rule is the appearance of 16 singleton byte candidates; the held-out generator key is consulted only afterward to validate the result. Every stage either verifies its prerequisites or reports that the instance is outside the demonstrated path.

The table factorizations are exact. Kernel recovery is probabilistic, with observable completion tests and the bounds in Eqs. (6)–(7). Conditional on full rank and label coverage, reconstruction is deterministic under the generator structure; key uniqueness follows from the finite AES check.

4.4 Effect of an Affine Input Offset

The role of a zero-preserving input decoder can be stated exactly. Consider a variation with the same table structure and injective output maps, but replace each selector $A_i x$ by $A_i x \oplus t_i$, for a

Table 2: Scope of the four key-recovery cohorts. Each contains 200 generated instances. AES results are from full encode–encrypt–decode block tests on those same instances; a dash means this cohort did not perform that test. The key comparison occurs only after the attack produces a candidate. Here q is the number of encoded inputs at first complete key recovery.

Cohort	AES functionality	Exact keys	Median/maximum q
General: direct AES check	200,000 matches	200/200	146/221
General: recovery cohort	—	200/200	146/174
Optimized: released C	199,800 mismatches in 200,000	200/200	146/210
Optimized: one-line repair	200,000 matches	200/200	146/160

fixed, unknown nibble t_i. The quotient label is then $f_i(x) = \pi_i(A_i x \oplus t_i)$. Equality of two labels is still equivalent to $A_i(x \oplus x') = 0$, so the recovered kernels, their observable rank conditions, and the sampling bounds in Eqs. (6)–(7) are unchanged. A recovered basis $B_i = M_i A_i$ still gives linear coordinates $u = B_i x$, but coordinate zero now corresponds to semantic nibble t_i.

The learned coordinate-to-label map includes this translation. Write k_j for the constant embedded in the first TypeII table at slot j; in the released generator it is the corresponding AES key byte. The reconstructed byte function becomes

$$\begin{aligned} V_j(u, v) &= H_j \mathsf{S}(k_j^{\text{eff}} \oplus L_{j,0} u \oplus L_{j,1} v) \oplus b_j, \\ k_j^{\text{eff}} &= k_j \oplus t_j, \end{aligned} \tag{16}$$

where t_j combines the two nibble offsets at first TypeII slot j after the initial ShiftRows permutation. The finite AES property therefore gives the singleton $\{k_j \oplus t_j\}$ for every byte key and every offset, under the same basis and output-map assumptions. Section 5.3 evaluates complete generator variants.

An isolated function in Eq. (16) cannot separate k_j from t_j: replacing both by their XOR with the same byte leaves the function unchanged. This does not by itself describe an AES-correct modified generator. Let D and $E = D^{-1}$ be the original external decoder and encoder, and let K be the original AES master key. Write K_j for its byte at first TypeII slot j after the initial ShiftRows, and t_j for the offset at that slot. If $D_t(x) = D(x) \oplus t$ is used while E and the TypeII constants are fixed, the plaintext entering the first round changes. Two compensations preserve AES with key K. Keeping E and changing the first TypeII constants to $K_j \oplus t_j$ leaves the S-box inputs unchanged, and Eq. (16) then identifies K_j. Alternatively, keeping the TypeII constants at K_j and changing the external encoder to $E_t(P) = E(P \oplus t)$ gives $D_t(E_t(P)) = P$, while the first-round oracle on chosen *encoded* inputs identifies $K_j \oplus t_j$. The key schedule in both cases remains that of K. These are different changes to the full implementation; an offset alone does not establish a secure repair. Section 5.3 tests both variants and whether adjacent-round relations identify K.

5 Evaluation

We evaluate whether the first-round attack recovers the exact master key of functioning AES instances, whether it extends to the released optimized C generator, what the nibble partition adds to affine recovery, and how the first-round and adjacent-round routes compare on identical targets. Table 2 summarizes the four key-recovery cohorts.

Each generated trial uses a new key and table set. The primary endpoint is the first encoded-input prefix at which the attack produces a unique full-key candidate, scored afterward against the held-out generator key.

5.1 Validation and Key Recovery on the General Source

Access and stopping rule The attack reads the 64 first-round TypeII tables, 256 first-stage TypeIV_II tables, and the public TypeIa/TypeIV_Ia input-layer tables of the pinned `WBSLT-general` generator. Each observation records a chosen encoded input and the 16 selector words before `ShiftRows`; the fixed permutation gives the actual TypeII indices. The general recovery cohort used uniform encoded inputs from the operating system's random source, and the cohort with direct AES checks used pseudorandom seeds fixed separately in advance. The attack receives no paired plaintexts or private encoder matrices. A separate evaluation file holds the AES key.

Each cohort contains 200 predeclared trials with no per-instance query ceiling. Beginning at $q = 140$, the first possible completion prefix, we test every prefix against the rank, label-coverage, and singleton conditions in Section 4.3. Inputs are added to the same instance until all 16 byte candidates are unique. The held-out key is used only after stopping; an incorrect singleton counts as failure.

Exact keys from functioning AES instances Each instance in the cohort with direct AES checks underwent 1,000 complete encode–encrypt–decode tests: zero, all ones, and 998 fresh random plaintexts. An independent AES implementation confirmed all 200,000 outputs. The input encoding and its inverse also recovered every tested plaintext. The attack used public tables exported from these same instances.

The attack recovered all 200 exact held-out keys. First successful prefixes ranged from 143 to 221 encoded inputs, with median 146. Every instance also had four address records from complete encryption, all of which matched the separate collector. Thus correctness and key extraction were checked on the same generated implementations.

Independent recovery cohort The separate general recovery cohort yielded 200/200 exact keys after 143–174 encoded inputs, with median 146. Independent recomputation from the raw records confirmed all 200 stopping prefixes and recovered keys.

Figure 3 shows both general cohorts and the two optimized-source cohorts. All endpoints are observed first recoveries under the common stopping rule.

Address collector validation Read-only GDB observations of the unmodified upstream `wbaes_encrypt` matched the collector on 3,456 inputs from 200 archived and four fresh instances: all 55,296 selector words and their TypeII indices agreed. The archived cases verify the first-round boundary; the fresh instances also completed encryption.

The two restriction tests provide another view of the recovered key. Figure 4 uses case 0 from the 200-instance general recovery cohort, at its first successful prefix ($q = 145$). At every AES key-byte position, each restriction retains an incorrect candidate, but their intersection contains only the true byte. The black markers form the recovered master key. The independent finite AES check in Section 4.3 establishes that the two candidate sets intersect only at the true key for all 256 possible byte keys under its stated affine-relation model.

5.2 Validation and Key Recovery on the Optimized C Generator

The optimized C generator at the same pinned commit has one 32-bit TypeII word per first-round byte slot rather than the general generator's four chunks. Each six-bit selector still passes through a permutation and the four-to-one map ψ in Eq. (4). Both the S-box and each byte's contribution to the MixColumns output are injective. The 32-bit column mixer is invertible, and each nibble encoding on the output is bijective. Equality across complete rows and columns therefore gives the same 16×16 quotient, as checks of the public tables verify. The first-stage tables remain disguised XOR, and the input decoder remains linear. The released source forms

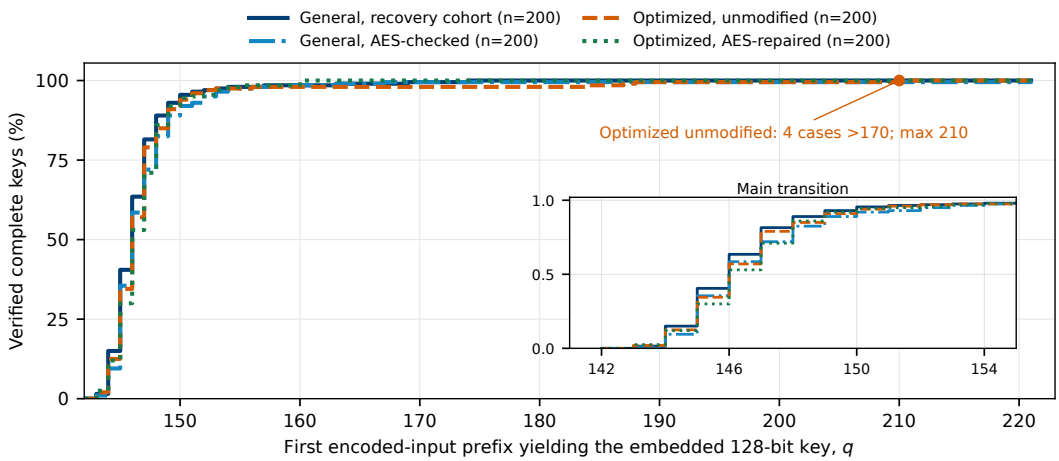

Figure 3: Observations needed for the first complete key recovery in four distinct 200-instance cohorts: the general recovery cohort, the separate general cohort checked against AES (200,000 blocks), the released WBSLT-optimized C source, and a separately labeled one-line repair that restores its AES behavior. Each step counts exact embedded 128-bit keys checked against the held-out generator key *after* the attack produced a unique candidate. The curves follow the uncapped stopping rule described in the text.

its first TypeII S-box input as $16\psi(lx)+\psi(rx)$, whereas the one-line repair uses $16\psi(rx)+\psi(lx)$. Our relation checker tests both orders. We evaluated separate, predeclared 200-instance cohorts of the released source and the version with only this nibble order corrected, using the same stopping and scoring rules.

The released optimized source failed its documented encode–encrypt–decode AES test on 199,800 of 200,000 blocks. The source with the one-line repair matched reference AES on all 200,000 tested blocks. These are separate targets: the first result concerns the released code's embedded key, while the second shows exact-key extraction from an instance of the same table construction that implements AES. The attack recovered all 200 held-out generator keys in each cohort. Both medians were 146 observations and both 95th percentiles were 151; the last observed prefixes were 210 for the released source and 160 for the repair.

Each recovered byte function passed checks on 64 fresh inputs per case. Across all 400 instances, an independent audit reconstructed stopping prefixes from raw address records, checked intersections of candidate sets, and rescored the saved candidates against the held-out generator keys.

5.3 Information Ablation on the Same Targets

For the *normalized first-round oracle alone*, the field-scaling symmetry of Section 4.2 gives at least 255^m compatible labels when m first-round key bytes are nonzero. In the general recovery cohort, 192 held-out keys have $m = 16$, seven have $m = 15$, and one has $m = 14$. This is an analytic ambiguity of the reduced oracle, not a claim about the full multi-round program.

We also fixed a separate 200-instance matched cohort before generation. For each new WBSLT target, an F-only analysis received values of the normalized first-round oracle derived from its public tables, without the row/column quotient. It recovered all 16 byte maps and constructed an affine-equivalent AES S-box representation for each. Every byte map passed a check at all 256 inputs, and 64 fresh inputs to the full oracle per instance passed a decomposition check after the run. The F-only analysis made 589–612 oracle calls for collection and its initial checks (median 599). On those *same* targets, the analysis with access to the nibble partition selected a unique 128-bit key after 143–171 queries that return addresses (median 146), and all

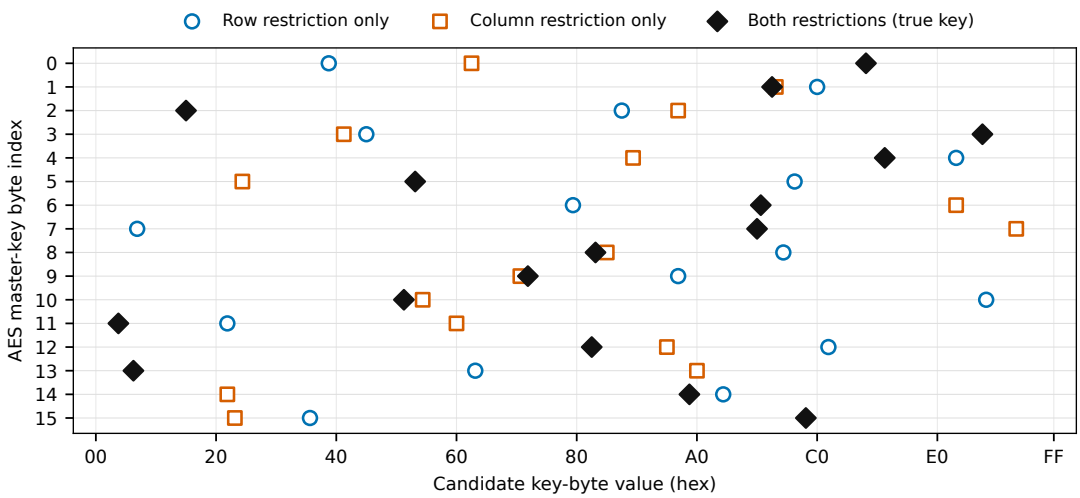

Figure 4: Exact candidate sets for each key byte in case 0 of the general recovery cohort, using its first successful prefix ($q = 145$). Open circles pass only the restriction to the row nibble; open squares pass only the restriction to the column nibble; each black diamond passes both and matches the held-out AES master-key byte.

200 candidates matched the held-out master keys after stopping. These call counts measure our F-only ablation and our exact-key attack using the nibble partition. They are not benchmarks of Derbez *et al.*'s Algorithm 1, which we did not run. That algorithm seeks an affine-equivalent representation, not necessarily the embedded WBSLT key.

All 200 F-only decompositions were reproduced in a process namespace that exposed only a socket accepting chosen inputs and returning values of F. The process had no access to tables, addresses, quotient labels, or evaluator keys, and each decomposition passed 64 additional checks. This validates the information boundary of the ablation, not exact-key recovery by F alone.

Adding the nibble spaces derived from the quotient to the F-only affine representatives gave one ordered fit within the self-equivalence family of Section 4.2 for each byte in all 200 cases and yielded the same exact keys as the direct attack. Candidates were saved before comparison. This holdout was procedural: the combined analysis and evaluator shared a Unix account.

Two-round alternative We separately fixed 20 WBSLT-general instances before generation to test whether relations across rounds can identify the embedded key without recovering the original nibble partition. All 20,000 complete block tests matched reference AES in an independent check. From public tables and first-round addresses, an adapter constructed normalized byte oracles F and G for the first two rounds. To evaluate G at arbitrary normalized first-round outputs, it selected raw TypeII addresses whose *joint* outputs agree exactly and evaluated the intervening TypeIV and TypeIII tables. It did not compute row or column classes or identify the decoder's original nibble partition. Both paths use exact public-table output equivalences, but only the first-round route identifies independent row and column classes. The adapter was checked on every TypeII address and first-stage TypeIV_II entry in both rounds, and against addresses recorded during complete encryption.

The two-round analysis decomposed both oracles into 16 byte maps, verified global input rank 128 and bijectivity of each byte map, and tested 128 fresh inputs against each decomposition. It enumerated the 2,040 known AES S-box self-equivalences per byte and matched candidate affine maps across all 64 nonzero byte blocks of the known ShiftRows–MixColumns relation. In every instance, one joint candidate within this enumerated family remained in each of the four MixColumns groups; all 20 AES-128 candidates matched the held-out keys.

This WBSLT-specific analysis follows the adjacent-round matching idea of Derbez *et al.* [5].

Table 3: Paired replay on the same 20 functioning WBSLT-general instances. Variable counts are medians [minimum, maximum]. Inference counts exclude validation; G uses public tables offline. Times include route-specific validation and describe only our implementations.

Measure	First round	Adjacent rounds
Exact keys	20/20	20/20
Live trace queries	147 [144, 233]	565.5 [558, 586]
Additional live checks	64	128
Offline G inference	—	567 [559, 576]
Public table bytes read	1,490,944	4,194,304
Median wall time	5.51 s	9.86 s

The paired comparison below measures both local implementations on the same targets.

Same-instance access and cost We replayed both routes on the fixed 20-instance cohort, using the same public exports and first-round address servers. Both recovered all 20 held-out keys. Neither used the reference address records supplied by the generator or evaluator key during inference. The two-round route computed G entirely from public tables; only its F evaluations requested first-round addresses for chosen encoded inputs. Table 3 separates these live evaluations from offline work and from later validation calls. Each live query submits one chosen encoded input and returns 16 first-round selector words.

The first-round count is the first prefix satisfying the observable stopping rule. The adjacent-round live count combines sampling with 256 evaluations that tabulate byte maps; its offline G count has the same components. The first route used 64 additional live checks, while the second used 128 live F and 128 offline G checks. We timed the routes sequentially on one Core Ultra 7 155H host, alternating their order. The reported times include public preprocessing, inference, and these route-specific checks; they exclude target generation, helper compilation, AES functionality tests, and final key scoring. These descriptive timings include different validation work and do not benchmark Derbez *et al.*'s implementation.

Output reachability at the composed boundary We examined the same 20 instances at the raw table boundary of the first round, from 16 twelve-bit addresses after `ShiftRows` through TypeII–TypeIV_II and TypeIII–TypeIV_III. Sampled outputs agreed with original C executions at the final boundary and with an independent Python evaluator after both combining layers. Holding 15 addresses at zero and varying address 0 through all 4,096 values produced only 256 distinct values of final output word 0 in every instance. The four-to-one reduction of each six-bit selector proves an upper bound of 256 for any fixed context and downstream output coordinate. Thus this restricted output image coexists with exact-key recovery. This is not a replay of the published BGE or MGH attacks; equivalence to their specified composed-function boundaries has not been established.

AES-preserving affine-offset variants We modified the pinned general generator at the last input-decoder merge, placing each offset before the private six-bit output permutation. The TypeII inverse permutation cancels that permutation before the selector map ψ; the identity $\psi(z \oplus (n \ll 2)) = \psi(z) \oplus n$ holds for all 64×16 selector–offset pairs. We fixed two disjoint 20-instance cohorts before generation and drew a fresh master key K and input offset t for each instance. Both variants kept the later-round key schedule of K. An independent AES implementation checked 1,000 complete encryptions per instance: all 20,000 blocks matched AES_K in each arm. The unchanged first-round and adjacent-round analyses received public tables and addresses observed for chosen encoded inputs; evaluator keys, offsets, and plaintext–encoded-input pairs were withheld.

Table 4: AES-preserving affine-offset variants. Each arm comprises 20 predeclared instances and 20,000 independently checked AES blocks. Entries show the unique first-round label and the adjacent-round master-key candidate.

Compensation	First round	Adjacent rounds
External encoder E_t	$K \oplus t$ (20/20)	K (20/20)
First TypeII constants	K (20/20)	K (20/20)

With the external encoder changed to E_t, the first-round route returned $K \oplus t$ in all 20 cases and never returned the original K. Using $K \oplus t$ as an AES master key matched none of the 20,000 block records. The adjacent-round route recovered K in all 20 cases. When the first TypeII constants instead compensated for the same decoder offset, both routes recovered K in all 20 cases. Thus the offset preserves the quotient and collision spaces, while the first-round key label depends on how AES functionality is maintained. The external encoder in the first variant was used only by the evaluator and withheld within the stated attack interface. A fully observing white-box adversary can inspect that encoder; neither variant is shown to be a secure countermeasure.

6 Discussion and Related Work

WBSLT shows why evaluations of designs with enlarged table addresses should inspect complete public tables for smaller selector partitions and determine what execution traces reveal about them.

WBSLT reports 10,000 unsuccessful trials of each of four adapted structural attacks: BGE, MGH, LRDRP, and BS [8]. Table 5 maps the predicates analyzed in the paper to the programs run by its released E4 scripts [9]. The original attack models include nibble encodings. The relevant difference is that these predicates concern a composed round function F or F'.

Table 5: WBSLT attack predicates and released E4 runner targets.

Attack	WBSLT equations and predicate; released E4 runner
BGE	Match with inferred inverse versus mismatch of composed F_0 (WBSLT Eqs. (13)–(14)); E4 uses a C++ generator with the partial adder disabled.
MGH	With other inputs fixed, F_i is bijective versus an unreachable output (WBSLT Eqs. (20)–(21)); the released E4 runner uses the same C++ generator and Python attack code as its BGE runner.
LRDRP	F_0 output match across input positions versus no match (WBSLT Eqs. (26)–(27)); E4 builds the optimized C generator studied here but tests a local `fprime` rather than its tables.
BS	Injectivity versus collision of composed F (WBSLT Eqs. (31)–(32)); E4 tests its own Go WBSLT generator.

The released BGE and MGH encryption runners compare the recovered value with a fresh random key rather than the fixed key embedded in their tables [9]. This success check does not measure recovery of the embedded table key.

Our quotient tests a different condition: equality of complete local TypeII rows, $T_j(a, b) = T_j(a', b)$ for *every* column b, and the analogous equality of columns. A failed match or a collision in a composed function does not rule out these local classes. None of the published predicates or released runners checks both the complete local equivalences and their relation to observed addresses. The released scripts document their own targets, not the exact code path of all 40,000 reported trials; they cannot identify the cause of each failure or provide same-instance

Table 6: Information used and recovery endpoints of the closest affine analyses.

Analysis	Information and result
Generic affine recovery [5]	Forward $B \circ S \circ A$ oracle; outputs equivalent affine layers, with no requirement to identify the original key label.
Baek key recovery [5]	Masked public tables and relations between two consecutive rounds; resolves self-equivalences and recovers the key in the Baek implementation.
Normalized WBSLT oracle	Public TypeII and XOR tables, together with addresses from the first round, provide such an oracle; its byte-level key labels have the affine symmetry of Section 4.2.
WBSLT two-round experiment (this work)	Public tables spanning two rounds and first-round addresses; adjacent-round matching recovers 20/20 exact keys without the nibble partition.
WBSLT first-round attack (this work)	Public tables and first-round addresses identify the original nibble partition; under the pinned linear decoder, two restrictions yield the exact master key.

cost baselines for our attack. Likewise, the analyzed cost of recovering the complete external encoding does not bound key extraction from the smaller structure used here.

At the measured 16-word raw boundary, the diagnostic on the same instances in Section 5.3 shows that an unreachable-output condition of the form in WBSLT Eq. (21) follows from the six-to-four-bit selector reduction that also underlies our TypeII quotient. The embedded key is still recovered through local table classes and addresses. This boundary has not been shown to reproduce the functions tested by the published BGE or MGH evaluations, so we do not infer their key-recovery success rates.

6.1 Relation to Prior Cryptanalysis

Our analysis builds on Chow-style table networks and their structural attacks [1,3,13,14]. Xiao–Lai encodings, perturbation, and dual-cipher representations illustrate earlier attempts to change the exposed AES structure, together with cryptanalysis of those changes [14–18]. Linear and affine equivalence and substitution–affine decomposition supply established tools for analyzing encoded functions [10, 11, 19].

Derbez *et al.*'s Algorithm 1 recovers an equivalent affine representation from a forward $B \circ S \circ A$ oracle [5]. Their attack on Baek *et al.*'s implementation [20] also recovers byte kernels from public tables and uses consecutive rounds to resolve S-box self-equivalences and identify the key. Our first-round route instead identifies WBSLT's original nibble partition; under the strictly linear decoder, its two S-box restrictions identify the key without an adjacent round. Table 6 distinguishes these recovery endpoints. Our adjacent-round WBSLT adaptation follows the prior matching idea, using our implementation rather than the published algorithm or Baek-specific code. Table 3 compares our two implementations on the same targets.

Differential computation analysis and grey-box attacks extract information from instrumented traces despite internal encodings [4, 21–24]. Higher-order, LPN-based, and filtering attacks study what remains observable under masking and shuffling [25–29]. Balanced encoding targets correlation leakage [30]; our quotient instead tests exact table equality and affine relations. Fault attacks on externally encoded AES use execution perturbations [31], while algebraic and automated BGE analyses expose structure in encoded computations [32, 33].

Other constructions change the primitive or security objective. Space-hard ciphers, WhiteBlock/WhiteKey, and SPNbox study incompressibility [34–36]. ASASA uses alternating affine and substitution layers [37]; implicit ARX implementations represent rounds by low-degree equations [38], and subsequent work gives attacks on those constructions [39]. ECDSA white-box designs extend the setting beyond symmetric encryption [40]. Applying our quotient criterion to

these families would require identifying a suitable public table boundary and establishing what its partitions reveal about the key.

Cryptanalysis of self-equivalence encodings also examines how algebraic structure limits encoding security [41]. Dummy shuffling targets algebraic attacks [42], masking targets computation and algebraic attacks [43–45], and table redundancy targets fault attacks [46]. Their security depends on the respective observation and intervention models, which must be assessed separately from WBSLT's exposed partitions.

Formal models distinguish key extraction from other white-box security goals [6, 7, 47]. Haystack ciphers model countermeasures as symmetric encryption, relating IND-CPA and IND-CCA security to computation-trace and fault attacks, respectively [48]. These distinctions motivate our explicit endpoint: recovery of the original AES key under the stated table and address access.

6.2 Design Implications and Limits

The general quotient criterion in Section 3.1 locates the structural issue in equality of complete observable rows and columns. A six-to-four-bit partial-adder map cannot be injective. Making all 64 input values distinguishable therefore requires changing how the surrounding network uses them, for example by retaining information that the current tables discard. Increasing table size alone leaves the quotient intact whenever the complete table still factors through the same smaller selector domains.

Replacing the linear boundary with a nonlinear encoding or changing the public XOR network could invalidate individual attack steps, but neither is a validated countermeasure. Affine offsets preserve the quotient and recovered row spaces; the adjacent-round route recovered the original key in both tested AES-preserving variants (Section 5.3). In the variant yielding $K \oplus t$ from one round, the modified external encoder is withheld from the attack interface, so this result does not establish protection under full local observation.

Evaluations should distinguish AES correctness, exact-key recovery, and recovery of a complete external encoding. They should test complete table equivalences, address collisions, and XOR factorization alongside statistical leakage and attack-suite results.

The results cover the pinned general and optimized generators under the access model of Section 2; other parameter sets and implementations require separate analysis. Clock-based table seeds do not sample all encodings independently and uniformly, so measured query counts are empirical costs rather than universal bounds. The exhaustive S-box result is confined to the recovered partition and linear-decoder assumptions.

7 Conclusion

Public table equivalences and first-round addresses expose WBSLT's original nibble partition, which identifies the embedded AES-128 key under the linear decoder. The attack recovered all keys in 200 general instances that passed 200,000 reference AES block tests, and also succeeded on the released and repaired optimized generators. On the paired targets, an adjacent-round route also recovered every key, while the first-round route used fewer live queries and less public table data. The adjacent-round route remained effective in both tested AES-preserving affine-offset variants. These results identify structural checks that evaluations of enlarged table designs should include: complete table equivalences, their relation to observed addresses, and the key information they preserve.

References

[1] S. Chow, P. Eisen, H. Johnson, and P. C. van Oorschot, "White-Box Cryptography and an AES Implementation," in *Selected Areas in Cryptography – SAC 2002*. Springer Berlin Heidelberg, 2003, pp. 250–270. [Online]. Available: https://doi.org/10.1007/3-540-36492-7_17

[2] ——, "A White-Box DES Implementation for DRM Applications," in *Digital Rights Management – DRM 2002*. Springer Berlin Heidelberg, 2003, pp. 1–15. [Online]. Available: https://doi.org/10.1007/978-3-540-44993-5_1

[3] O. Billet, H. Gilbert, and C. Ech-Chatbi, "Cryptanalysis of a White Box AES Implementation," in *Selected Areas in Cryptography – SAC 2004*. Springer Berlin Heidelberg, 2004, pp. 227–240. [Online]. Available: https://doi.org/10.1007/978-3-540-30564-4_16

[4] J. W. Bos, C. Hubain, W. Michiels, and P. Teuwen, "Differential Computation Analysis: Hiding Your White-Box Designs is Not Enough," in *Cryptographic Hardware and Embedded Systems – CHES 2016*. Springer Berlin Heidelberg, 2016, pp. 215–236. [Online]. Available: https://doi.org/10.1007/978-3-662-53140-2_11

[5] P. Derbez, P.-A. Fouque, B. Lambin, and B. Minaud, "On Recovering Affine Encodings in White-Box Implementations," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2018, no. 3, pp. 121–149, 2018. [Online]. Available: https://doi.org/10.46586/tches.v2018.i3.121-149

[6] C. Delerablée, T. Lepoint, P. Paillier, and M. Rivain, "White-Box Security Notions for Symmetric Encryption Schemes," in *Selected Areas in Cryptography – SAC 2013*. Springer Berlin Heidelberg, 2014, pp. 247–264. [Online]. Available: https://doi.org/10.1007/978-3-662-43414-7_13

[7] E. Alpirez Bock, A. Amadori, C. Brzuska, and W. Michiels, "On the Security Goals of White-Box Cryptography," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2020, no. 2, pp. 327–357, 2020. [Online]. Available: https://doi.org/10.46586/tches.v2020.i2.327-357

[8] Y. Shi, T. Gao, Y. Li, J. Gao, and K. Huang, "WBSLT: A Framework for White-Box Encryption Based on Substitution-Linear Transformation Ciphers," in *Network and Distributed System Security Symposium – NDSS 2026*. Internet Society, 2026. [Online]. Available: https://doi.org/10.14722/ndss.2026.242492

[9] T. Gao, "Artifacts of Research Paper "WBSLT: A Framework for White-Box Encryption Based on Substitution-Linear Transformation Ciphers", for NDSS 2026 Conference," 2025. [Online]. Available: https://doi.org/10.5281/zenodo.17543341

[10] A. Biryukov, C. De Cannière, A. Braeken, and B. Preneel, "A Toolbox for Cryptanalysis: Linear and Affine Equivalence Algorithms," in *Advances in Cryptology – EUROCRYPT 2003*. Springer Berlin Heidelberg, 2003, pp. 33–50. [Online]. Available: https://doi.org/10.1007/3-540-39200-9_3

[11] A. Biryukov and A. Shamir, "Structural Cryptanalysis of SASAS," in *Advances in Cryptology – EUROCRYPT 2001*. Springer Berlin Heidelberg, 2001, pp. 395–405. [Online]. Available: https://doi.org/10.1007/3-540-44987-6_24

[12] National Institute of Standards and Technology, "Advanced Encryption Standard (AES)," National Institute of Standards and Technology, Tech. Rep. FIPS 197-upd1, 2023. [Online]. Available: https://csrc.nist.gov/pubs/fips/197/final

[13] W. Michiels, P. Gorissen, and H. D. L. Hollmann, "Cryptanalysis of a Generic Class of White-Box Implementations," in *Selected Areas in Cryptography – SAC 2008*. Springer Berlin Heidelberg, 2009, pp. 414–428. [Online]. Available: https://doi.org/10.1007/978-3-642-04159-4_27

[14] T. Lepoint, M. Rivain, Y. De Mulder, P. Roelse, and B. Preneel, "Two Attacks on a White-Box AES Implementation," in *Selected Areas in Cryptography – SAC 2013*. Springer Berlin Heidelberg, 2014, pp. 265–285. [Online]. Available: https://doi.org/10.1007/978-3-662-43414-7_14

[15] Y. Xiao and X. Lai, "A Secure Implementation of White-Box AES," in *2009 2nd International Conference on Computer Science and its Applications*. IEEE, 2009, pp. 1–6. [Online]. Available: https://doi.org/10.1109/csa.2009.5404239

[16] Y. De Mulder, P. Roelse, and B. Preneel, "Cryptanalysis of the Xiao – Lai White-Box AES Implementation," in *Selected Areas in Cryptography – SAC 2012*. Springer Berlin Heidelberg, 2013, pp. 34–49. [Online]. Available: https://doi.org/10.1007/978-3-642-35999-6_3

[17] Y. De Mulder, B. Wyseur, and B. Preneel, "Cryptanalysis of a Perturbated White-Box AES Implementation," in *Progress in Cryptology - INDOCRYPT 2010*. Springer Berlin Heidelberg, 2010, pp. 292–310. [Online]. Available: https://doi.org/10.1007/978-3-642-17401-8_21

[18] M. Karroumi, "Protecting White-Box AES with Dual Ciphers," in *Information Security and Cryptology - ICISC 2010*. Springer Berlin Heidelberg, 2011, pp. 278–291. [Online]. Available: https://doi.org/10.1007/978-3-642-24209-0_19

[19] I. Dinur, "An Improved Affine Equivalence Algorithm for Random Permutations," in *Advances in Cryptology – EUROCRYPT 2018*. Springer International Publishing, 2018, pp. 413–442. [Online]. Available: https://doi.org/10.1007/978-3-319-78381-9_16

[20] C. H. Baek, J. H. Cheon, and H. Hong, "White-box AES implementation revisited," *Journal of Communications and Networks*, vol. 18, no. 3, pp. 273–287, 2016. [Online]. Available: https://doi.org/10.1109/jcn.2016.000043

[21] E. Alpirez Bock, C. Brzuska, W. Michiels, and A. Treff, "On the Ineffectiveness of Internal Encodings - Revisiting the DCA Attack on White-Box Cryptography," in *Applied Cryptography and Network Security*. Springer International Publishing, 2018, pp. 103–120. [Online]. Available: https://doi.org/10.1007/978-3-319-93387-0_6

[22] M. Rivain and J. Wang, "Analysis and Improvement of Differential Computation Attacks against Internally-Encoded White-Box Implementations," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2019, no. 2, pp. 225–255, 2019. [Online]. Available: https://doi.org/10.46586/tches.v2019.i2.225-255

[23] E. Alpirez Bock, J. W. Bos, C. Brzuska, C. Hubain, W. Michiels, C. Mune, E. Sanfelix Gonzalez, P. Teuwen, and A. Treff, "White-Box Cryptography: Don't Forget About Grey-Box Attacks," *Journal of Cryptology*, vol. 32, no. 4, pp. 1095–1143, 2019. [Online]. Available: https://doi.org/10.1007/s00145-019-09315-1

[24] Y. Tang, Z. Gong, B. Li, and L. Zhao, "Revisiting the Computation Analysis against Internal Encodings in White-Box Implementations," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2023, no. 4, pp. 493–522, 2023. [Online]. Available: https://doi.org/10.46586/tches.v2023.i4.493-522

[25] A. Bogdanov, M. Rivain, P. S. Vejre, and J. Wang, "Higher-Order DCA against Standard Side-Channel Countermeasures," in *Constructive Side-Channel Analysis and Secure Design – COSADE 2019*. Springer International Publishing, 2019, pp. 118–141. [Online]. Available: https://doi.org/10.1007/978-3-030-16350-1_8

[26] L. Goubin, M. Rivain, and J. Wang, "Defeating State-of-the-Art White-Box Countermeasures with Advanced Gray-Box Attacks," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2020, no. 3, pp. 454–482, 2020. [Online]. Available: https://doi.org/10.46586/tches.v2020.i3.454-482

[27] Y. Tang, Z. Gong, J. Chen, and N. Xie, "Higher-Order DCA Attacks on White-Box Implementations with Masking and Shuffling Countermeasures," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2023, no. 1, pp. 369–400, 2023. [Online]. Available: https://doi.org/10.46586/tches.v2023.i1.369-400

[28] A. Charlès and A. Udovenko, "LPN-based Attacks in the White-box Setting," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2023, no. 4, pp. 318–343, 2023. [Online]. Available: https://doi.org/10.46586/tches.v2023.i4.318-343

[29] ——, "White-box filtering attacks breaking SEL masking: from exponential to polynomial time," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2024, no. 3, pp. 1–24, 2024. [Online]. Available: https://doi.org/10.46586/tches.v2024.i3.1-24

[30] S. Lee and J.-N. Kim, "Balanced Encoding of Near-Zero Correlation for an AES Implementation," *IEEE Transactions on Information Forensics and Security*, vol. 19, pp. 6589–6603, 2024. [Online]. Available: https://doi.org/10.1109/tifs.2024.3420101

[31] A. Amadori, W. Michiels, and P. Roelse, "A DFA Attack on White-Box Implementations of AES with External Encodings," in *Selected Areas in Cryptography – SAC 2019*. Springer International Publishing, 2020, pp. 591–617. [Online]. Available: https://doi.org/10.1007/978-3-030-38471-5_24

[32] A. Biryukov and A. Udovenko, "Attacks and Countermeasures for White-box Designs," in *Advances in Cryptology – ASIACRYPT 2018*. Springer International Publishing, 2018, pp. 373–402. [Online]. Available: https://doi.org/10.1007/978-3-030-03329-3_13

[33] A. Amadori, W. Michiels, and P. Roelse, "Automating the BGE Attack on White-Box Implementations of AES with External Encodings," in *2020 IEEE 10th International Conference on Consumer Electronics (ICCE-Berlin)*. IEEE, 2020, pp. 1–6. [Online]. Available: https://doi.org/10.1109/icce-berlin50680.2020.9352195

[34] A. Bogdanov and T. Isobe, "White-Box Cryptography Revisited: Space-Hard Ciphers," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015, pp. 1058–1069. [Online]. Available: https://doi.org/10.1145/2810103.2813699

[35] P.-A. Fouque, P. Karpman, P. Kirchner, and B. Minaud, "Efficient and Provable White-Box Primitives," in *Advances in Cryptology – ASIACRYPT 2016*. Springer Berlin Heidelberg, 2016, pp. 159–188. [Online]. Available: https://doi.org/10.1007/978-3-662-53887-6_6

[36] A. Bogdanov, T. Isobe, and E. Tischhauser, "Towards Practical Whitebox Cryptography: Optimizing Efficiency and Space Hardness," in *Advances in Cryptology – ASIACRYPT 2016*. Springer Berlin Heidelberg, 2016, pp. 126–158. [Online]. Available: https://doi.org/10.1007/978-3-662-53887-6_5

[37] A. Biryukov, C. Bouillaguet, and D. Khovratovich, "Cryptographic Schemes Based on the ASASA Structure: Black-Box, White-Box, and Public-Key (Extended Abstract)," in *Advances in Cryptology – ASIACRYPT 2014*. Springer Berlin Heidelberg, 2014, pp. 63–84. [Online]. Available: https://doi.org/10.1007/978-3-662-45611-8_4

[38] A. Ranea, J. Vandersmissen, and B. Preneel, "Implicit White-Box Implementations: White-Boxing ARX Ciphers," in *Advances in Cryptology – CRYPTO 2022*. Springer Nature Switzerland, 2022, pp. 33–63. [Online]. Available: https://doi.org/10.1007/978-3-031-15802-5_2

[39] A. Biryukov, B. Lambin, and A. Udovenko, "Cryptanalysis of ARX-based White-box Implementations," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2023, no. 3, pp. 97–135, 2023. [Online]. Available: https://doi.org/10.46586/tches.v2023.i3.97-135

[40] G. Barbu, W. Beullens, E. Dottax, C. Giraud, A. Houzelot, C. Li, M. Mahzoun, A. Ranea, and J. Xie, "ECDSA White-Box Implementations: Attacks and Designs from CHES 2021 Challenge," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2022, no. 4, pp. 527–552, 2022. [Online]. Available: https://doi.org/10.46586/tches.v2022.i4.527-552

[41] A. Ranea and B. Preneel, "On Self-equivalence Encodings in White-Box Implementations," in *Selected Areas in Cryptography – SAC 2020*. Springer International Publishing, 2021, pp. 639–669. [Online]. Available: https://doi.org/10.1007/978-3-030-81652-0_25

[42] A. Biryukov and A. Udovenko, "Dummy Shuffling Against Algebraic Attacks in White-Box Implementations," in *Advances in Cryptology – EUROCRYPT 2021*. Springer International Publishing, 2021, pp. 219–248. [Online]. Available: https://doi.org/10.1007/978-3-030-77886-6_8

[43] S. Lee, T. Kim, and Y. Kang, "A Masked White-Box Cryptographic Implementation for Protecting Against Differential Computation Analysis," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 10, pp. 2602–2615, 2018. [Online]. Available: https://doi.org/10.1109/tifs.2018.2825939

[44] S. Lee and M. Kim, "Improvement on a Masked White-Box Cryptographic Implementation," *IEEE Access*, vol. 8, pp. 90 992–91 004, 2020. [Online]. Available: https://doi.org/10.1109/access.2020.2993651

[45] O. Seker, T. Eisenbarth, and M. Liśkiewicz, "A White-Box Masking Scheme Resisting Computational and Algebraic Attacks," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2021, no. 2, pp. 61–105, 2021. [Online]. Available: https://doi.org/10.46586/tches.v2021.i2.61-105

[46] S. Lee, N.-S. Jho, and M. Kim, "Table Redundancy Method for Protecting Against Fault Attacks," *IEEE Access*, vol. 9, pp. 92 214–92 223, 2021. [Online]. Available: https://doi.org/10.1109/access.2021.3092314

[47] A. Saxena, B. Wyseur, and B. Preneel, "Towards Security Notions for White-Box Cryptography," in *Information Security – ISC 2009*. Springer Berlin Heidelberg, 2009, pp. 49–58. [Online]. Available: https://doi.org/10.1007/978-3-642-04474-8_4

[48] A. Charlès and A. Udovenko, "Haystack Ciphers: White-Box Countermeasures as Symmetric Encryption," in *Advances in Cryptology – ASIACRYPT 2025*. Springer Nature Singapore, 2026, pp. 341–374. [Online]. Available: https://doi.org/10.1007/978-981-95-5096-8_11

A Ethical Considerations

This study analyzes a publicly released academic implementation using locally generated instances. It involves no third-party deployment, user data, network probing, or claims about operational systems. The result provides a reproducible test of a published claim about resistance to key extraction, limited to the pinned upstream source. Publication of the method could aid key extraction from implementations derived from the affected generator; their existence and use are not established by this study. We weighed that possible harm against the value of correcting the security claim and enabling independent evaluation. We privately notified the WBSLT authors on November 2, 2026, and provided a technical summary of our findings. We will assess potential downstream use and coordinate any broader dissemination of the code with the WBSLT authors.