Table-Free Constant-Work BCH Decoding by Algebraic Index Elimination

Vincent Cheng[1], Zhaohui Cheng[2], Junfeng Fan[3] and Kaixin Xiong[2]

[1] Northwestern University, v.z.cheng@hotmail.com
[2] Shenzhen Olym Info. Sec. Ltd, chengzh@myibc.net, xiongkx@myibc.net
[3] Open Security Research, fan@osr-tech.com

Abstract. Algebraic error-correcting codes such as BCH let lattice-based key-encapsulation mechanisms (KEMs) reduce bandwidth and the decryption-failure rate. The decoder's input is determined jointly by the long-term secret and a ciphertext that an adversary may choose. Variable work or table accesses that reveal error positions can enable key recovery even when the KEM uses implicit rejection. Walters and Sinha Roy (WSR) gave a constant-work BCH decoder whose fixed-address realization scans the entire field per protected lookup.

We introduce *algebraic index elimination*, a table-free realization of WSR's paired Berlekamp–Massey schedule in which received-word-dependent selections become fixed circuits over public addresses. We prove bounded-distance correctness through radius t and source-level trace regularity for the parameterized decoder. We instantiate the construction with length-511 subcodes of dimension k, using $(k, t) = (384, 14)$ and $(320, 22)$. Their implementations eliminate, respectively, 5.2 and 7.2 KiB of received-word-selected field and encoder tables required by matched fixed-address WSR baselines. On WSR's published parameter sets, an Intel i7-13700 runs the deterministic cores 35.7–82.4× faster than the fixed-address WSR baselines. In the identity-based lattice KEMs ANSA-IBKEM1 and ANSA-IBKEM2, decapsulation is 25.5× and 10.9× faster than with fixed-address WSR baselines for the same 384-dimensional subcode. Index elimination does not equalize operands. Ordinary C masks compiled to received-word-dependent branches under Apple Clang 21; a compiler-opaque primitive prevented this transformation in the audited builds. At the measured radii, scalar binaries of both codes pass the x86-64, Cortex-A76 and Cortex-M4 mean-timing screens, and the Apple M5 screens with Data Independent Timing (DIT) enabled.

Keywords: BCH codes · constant-work cryptography · side-channel analysis · compiler security · post-quantum cryptography

1 Introduction

Lattice key-encapsulation mechanisms (KEMs) may employ an algebraic error-correcting code to tolerate additional decryption noise, thereby reducing bandwidth or the decryption-failure rate. The decoder then becomes a chosen-ciphertext interface: its input is a received word determined jointly by the long-term secret and a ciphertext that an adversary may choose. A work profile that depends on decoding success, error weight, or error position can expose a decryption-failure or error-location oracle even when the KEM hides the decoder's Boolean result by implicit rejection. Error positions are particularly damaging, because they relate an adversarial perturbation to a secret-dependent coordinate [DVV18, GNJ26].

This interface arises in several KEM designs. LAC used many-error BCH to support a byte-level modulus [LLZ+18]; ThreeBears used a Melas BCH code to increase noise, and

hence estimated security, within a fixed ciphertext budget [Ham19]; ANSA-IBKEM [CX26] uses length-511 BCH-derived subcodes in a lattice identity-based KEM. An earlier HQC design, which is code-based rather than lattice-based, likewise used BCH; current HQC does not [AMAB+20]. Timing attacks recovered LAC keys from variable-time BCH decoding [DTVV19], and power analysis recovered keys from the earlier HQC BCH decoder [SRSWZ21]. After HQC replaced BCH by Reed–Muller (RM) plus Reed–Solomon (RS), Goy et al. gave a single-trace soft analytical side-channel attack (SASCA) on the RS encoder and decoder [GMGL24]. Paiva et al. gave a shuffle-tolerant side-channel-assisted chosen-ciphertext attack on the RM decoder [PRJ+23]. Changing the code family does not remove the decoder as a security boundary.

Binary BCH codes can carry a few hundred information bits while correcting up to a chosen number t of bit errors. The complexity of constant-time decoding has nevertheless led some KEM designs to limit error correction or choose other code families. Hamburg rejected a six-error BCH in ThreeBears because "decoding many errors in constant time is rather complex", retaining a two-error Melas code that avoids a constant-time Berlekamp–Massey step [Ham19]. PolarLAC replaced LAC's BCH by polar codes; one reason given for that move is that polar codes support constant-time implementation [LWC+26].

Walters and Sinha Roy (WSR) exhibited a paired constant-work Berlekamp–Massey schedule (ConSiBMA) that exploits the Frobenius relation between binary BCH syndromes, processing two Newton identities in each of t fixed rounds [WSR20]. Public loop bounds and conditional assignment already remove the usual control-flow variation. What remains is an array index that depends on the received word. WSR's fixed-address method, which they call *blinded array access*, forms that index and then scans every table entry, masking the desired value; we call this *masked-table selection* to distinguish deterministic selection masks from probabilistic blinding. Their faster cache-preload alternative still ends with a received-word-dependent (RWD) access. WSR's fixed-address realization over $\mathbb{F}_{2^9}$ therefore examines all 512 field elements for every protected lookup [WSR20]. However, the tables increase the decoder's read-only storage footprint (Table 3), and these full-domain scans dominate WSR's decoding work at the evaluated parameters (Proposition 2 and Section 4.4).

Algebraic index elimination replaces these scans with circuits over public addresses. We retain WSR's current locator and selected predecessor, express each polynomial shift as a sum over public destination–source coefficient pairs, and compute field products with fixed polynomial-basis rounds. The RWD shift amount controls equality masks without forming an address. Encoding, syndrome evaluation, Chien search, correction, and candidate/status production likewise follow fixed schedules. The construction is parameterized by the code length $n = 2^m - 1$ and correction radius t. We call the implementation *table-free* because field operations on RWD values use arithmetic circuits rather than RWD-indexed lookups or scan-and-select over field tables. Publicly addressed constants and the eight encoder basis rows remain. For $m = 9$, we specify two length-511 subcodes, $\mathcal{C}_{384}$ and $\mathcal{C}_{320}$, with $(k, t) = (384, 14)$ and $(320, 22)$, respectively. We evaluate both codes and WSR's two published shortened codes; ANSA-IBKEM1 and ANSA-IBKEM2 deploy $\mathcal{C}_{384}$ and ANSA-IBKEM1536 deploys $\mathcal{C}_{320}$.

Table 1 summarizes the changes relative to WSR's fixed-address realization. Fixed control flow and addresses still permit class-dependent operand values: clean words have zero syndromes and many zero intermediates. On Apple M5, the table-free backend exhibits distinguishable timing with Data Independent Timing (DIT) disabled (Section 4.3).

Compiler preservation and operand timing therefore require separate checks (Sections 3.4 and 4.3). Whether DIT is needed depends on the compiled binary and processor; the source-level trace argument makes no DIT assumption.

Contributions. Our contributions are as follows.

Table 1: The fixed-address WSR baseline and our $m = 9$ realization. Both use WSR's t-round paired recurrence, degree rule, and complete Chien traversal over all 511 points. Both target received-word-independent control flow and memory addresses. Decoder and KEM latencies appear in Tables 7 and 9.

Property	Fixed-address WSR baseline	This work
Field arithmetic and Chien evaluation	Log/antilog values selected by scanning all 512 entries	Polynomial-basis arithmetic: nine-round multiplication, fixed squaring and inversion; Chien terms multiplied by public α^j
Locator updates	Masked reads and writes at RWD coefficient indices	The same carried predecessor; locator shifts evaluated over all public destination–source pairs
Encoding and syndromes	256-row masked remainder selection; syndromes from the parity discrepancy	Eight public encoder basis rows; syndromes from the same parity discrepancy
Correction writes	Masked writes to root slots; masked data reads and writes selected by error locations	Fixed writes at public coordinates, controlled by root masks

1. We give a parameterized table-free realization of WSR's paired BCH schedule by algebraic index elimination. Polynomial-basis field circuits, public coefficient-pair relations, and an eight-element encoder basis replace received-word-dependent table selections throughout decoding. We prove bounded-distance correctness and source-level trace regularity for the parameterized construction, covering correction, failure, and cleanup. The cost analysis separates WSR's full-field scans from field arithmetic and coefficient-pair operations.

2. We identify concrete compiler and operand-timing effects in the evaluated implementations. Apple Clang 21 converts ordinary C masks into received-word-dependent branches; a compiler-opaque primitive prevents this transformation in the audited binaries. On Apple M5, both evaluated scalar decoders exhibit timing differences with DIT disabled despite fixed control flow and addresses. These findings establish implementation-specific limits on timing qualification; the microarchitectural cause remains unknown.

3. We quantify performance and resource costs against fixed-address WSR baselines at matched code parameters and decoding boundaries. Decoder measurements characterize the speed, table-space, code-size, and stack tradeoffs, while ANSA-IBKEM integration measures the effect on complete decapsulation. The comparisons evaluate complete realizations without attributing the speedup to individual transformations or ranking physical side-channel resistance.

2 Background and threat model

2.1 Codes and decoding coordinates

Let $n = 2^m - 1$ and let α generate the multiplicative group $\mathbb{F}_{2^m}^*$. We write $\mathrm{BCH}_{\mathrm{ns}}(n, k, \delta)$ for the binary BCH code of dimension k and designed distance δ with defining roots $\alpha, \alpha^2, \ldots, \alpha^{\delta-1}$. The code is called *primitive* because its length is $2^m - 1$, and *narrow-sense* because the root sequence starts at α [HP03]. Its generator g_δ is the least common multiple of the minimal polynomials of these roots, and its minimum distance is at least δ. A decoder may use any positive radius $t \le \lfloor (\delta - 1)/2 \rfloor$. A linear subcode retains these roots and the corresponding bounded-distance decoder, even when the subcode itself is not cyclic.

Fix such a BCH parent $\mathcal{B}$ and a decoding radius t within its designed bound. A k-dimensional systematic subcode is specified by a public linear parity map $P : \mathbb{F}_2^k \to \mathbb{F}_2^{n-k}$,

whose encoder image satisfies

$$\mathcal{C}_P = \{(a, P(a)) : a \in \mathbb{F}_2^k\} \subseteq \mathcal{B}. \tag{1}$$

For a transmitted codeword $c \in \mathcal{C}_P$, write the received word as $y = c + e$ with $e \in \mathbb{F}_2^n$. A public bijection $\pi : \{0, \ldots, n-1\} \to \mathbb{Z}_n$ assigns serialized position q to the coefficient of $x^{\pi(q)}$ in the polynomial representation. The decoder computes $S_j = y(\alpha^j) = e(\alpha^j)$ for $1 \le j \le 2t$. For error support $E = \{q : e_q \neq 0\}$ of weight $w = |E| \le t$, the locator $\Lambda_E(X) = \prod_{q \in E}(1 - X\alpha^{\pi(q)})$ has degree w and root $\alpha^{-\pi(q)}$ for each error position q.

A Berlekamp–Massey-type recurrence recovers this locator from the syndromes [Mas69]. Chien search evaluates it over the nonzero field elements to find the error positions [Chi64]; correcting them recovers c. The locator degree, discrepancy sequence, and roots depend on the error pattern. Cho and Sung describe an optimized software realization of systematic BCH encoding and decoding [CS09].

Our concrete instances use $m = 9$ and $n = 511$. The 384-data-bit code used in ANSA-IBKEM is derived from $\mathcal{B}_{29} = \mathrm{BCH}_{\mathrm{ns}}(511, 385, 29)$ with $\deg g_{29} = 126$. Since the narrow-sense root set does not contain $\alpha^0 = 1$, the factor $x + 1$ is absent from $g_{29}(x)$. Consequently,

$$\mathcal{C}_{384} = \big\langle (x+1) g_{29}(x) \big\rangle \tag{2}$$

is a 384-dimensional even-weight cyclic subcode of $\mathcal{B}_{29}$. Its generator has degree 127 and divides $x^{511} - 1$. Even weight strengthens the BCH distance bound to $d(\mathcal{C}_{384}) \ge 30$. The roots inherited from $\mathcal{B}_{29}$ support bounded-distance decoding through radius $t = 14$.

The 320-data-bit code is derived from $\mathcal{B}_{45} = \mathrm{BCH}_{\mathrm{ns}}(511, 322, 45)$, whose generator g_{45} has degree 189. The binary cyclotomic cosets meeting exponents $1, \ldots, 44$ have total size 189. Adjoining $x+1$ gives the even-weight cyclic code $\mathcal{C}_{321} = \langle (x+1) g_{45}(x) \rangle$, with distance at least 46. Multiplying by this factor once more defines

$$\mathcal{C}_{320} = \big\{ a(x)(x+1)^2 g_{45}(x) : \deg a < 320 \big\} \subset \mathbb{F}_2[x]_{<511}. \tag{3}$$

The degree-191 polynomial $(x+1)^2 g_{45}$ gives dimension 320 without removing any coordinate; $\mathcal{C}_{320}$ is neither shortened nor punctured. It is noncyclic: its monic codeword of least degree is $(x+1)^2 g_{45}$, which cannot generate a cyclic code because it does not divide $x^{511} - 1$. It remains a subcode of $\mathcal{C}_{321}$, so $d(\mathcal{C}_{320}) \ge 46$. The decoder uses $S_1, \ldots, S_{44}$ and radius $t = 22$ through $\mathcal{B}_{45}$.

At the same length, $\mathcal{C}_{320}$ trades 64 data bits for 64 additional parity bits and eight additional correctable errors. The enclosing system chooses this rate–radius tradeoff through the public compile-time parameters (k, t). Both codes are decoded through their cyclic parents, a relationship that also supports the optional syndrome-domain shift in Section 3.6.

For these instances, serialize the 511 logical positions as $q = 0, \ldots, 510$, with the k systematic positions followed by the $r = 511 - k$ parity positions. Position q represents polynomial exponent

$$\pi(q) = -(q+1) \pmod{511}. \tag{4}$$

Thus parity-local position h has exponent $\pi(k+h) = r - 1 - h$, and the Chien point for position q is $\alpha^{-\pi(q)} = \alpha^{q+1}$. The public storage layout uses 64 bytes: 511 bits represent code coordinates, and the final bit is reserved and excluded from coordinate rotations. The API requires zero padding in unused systematic positions.

2.2 Adversary and leakage levels

An adversary can submit adaptively chosen ciphertexts for decapsulation under a fixed long-term key. We distinguish four leakage levels.

Protocol output. The decoder's validity bit must remain internal to the KEM's re-encryption and implicit-rejection logic. Return codes and response lengths must not expose it. The KEM's ciphertext comparison and shared-key selection must execute in constant time.

Timing/control. The attacker observes total or local execution time, branch behavior, or memory-access patterns. Our primary target is a received-word-independent control-flow and memory-address trace for each public input length.

Power/EM. A local attacker observes leakage correlated with data values, for example $L(x) = a\mathsf{HW}(x) + N$. Such leakage is outside the primary security claim. We describe an optional stage-local coordinate-randomization heuristic in Section 3.6, but claim neither probing security nor measured power/EM resistance.

Faults. Attacks that skip decoding, fix randomness, or corrupt the success bit are outside the present scope.

Fix the public decoder configuration, a valid input length, and the code and data base addresses. For a received word y, represent the deterministic decoder's control-flow and address trace as the ordered event sequence $\mathsf{Trace}_{\mathrm{ca}}(y) = (\varepsilon_1, \ldots, \varepsilon_N)$, where each ε_h is an instruction fetch $\mathsf{I}(p, \nu)$ or a ν-byte load, store, or read–modify–write $\mathsf{L}(a, \nu)$, $\mathsf{S}(a, \nu)$, $\mathsf{M}(a, \nu)$. Events retain their dynamic order, including the interleaving of instruction and memory events. Our pointwise objective is

$$\mathsf{Trace}_{\mathrm{ca}}(y_0) = \mathsf{Trace}_{\mathrm{ca}}(y_1), \tag{5}$$

for all received words y_0, y_1 of the fixed public length, including words that fail decoding or public-format checks. This includes changes in error count, position, candidate data, and decoder success. Public pointer, length, and configuration checks may retain early returns because the objective assumes fixed, valid arguments. Within that context, it excludes RWD architectural branches and effective addresses. It does not constrain data values, register contents, elapsed time, physical leakage, transient execution, or the processor's full microarchitectural state.

Timing independence additionally requires operand-independent instruction timing. The experiments in Section 4.3 screen the evaluated binaries and input classes for measurable timing dependence; passing those tests does not prove timing noninterference.

Conventional optimizations expose timing/control leakage by returning early on zero syndromes; skipping locator updates on zero discrepancies; bounding loops by the actual locator degree; stopping Chien search after the expected number of roots; writing only at root locations; taking distinct failure paths for inconsistent degree and root counts or invalid padding; or returning or re-encoding only a successful candidate. These input-dependent branches and accesses violate Equation (5).

For the optional position-randomizing wrapper of Section 3.6, let $\omega = (u, \rho) \in \mathbb{F}_2^k \times \mathbb{Z}_n$ denote the random message mask and cyclic coordinate shift. Using the same event model, its protected-call trace must satisfy

$$\mathsf{Trace}_{\mathrm{ca}}^{\mathrm{wrap}}(y_0; \omega_0) = \mathsf{Trace}_{\mathrm{ca}}^{\mathrm{wrap}}(y_1; \omega_1)$$

for all received words of the fixed public length and all valid coin strings in the fixed public context. This extends the deterministic objective in Equation (5) to both inputs. Entropy acquisition lies outside the protected call. Its rejection sampling depends only on random bytes, never on y, so including it preserves equality in distribution.

3 Table-free decoding by algebraic index elimination

We first formulate the indexed operations required by WSR's paired recurrence, then give their algebraic replacements and prove correctness and source-level trace regularity. We next address compiler preservation, analyze implementation costs, and describe the optional position-randomizing wrapper.

3.1 Indexed operations in WSR's paired decoder

WSR's ConSiBMA uses binary syndrome conjugacy to process two Newton identities in each of t public rounds. At the start of round $i = 0, \ldots, t-1$, the current locator $\Lambda^{(i)}$ satisfies the first $2i$ identities. Writing $\ell_i = \deg \Lambda^{(i)}$, its odd discrepancy is

$$d_i = S_{2i+1} + \sum_{j=1}^{\ell_i} \Lambda_j^{(i)} S_{2i+1-j}. \tag{6}$$

For nonzero d_i, the paired update is

$$\Lambda^{(i+1)}(X) = \Lambda^{(i)}(X) + d_i d_B^{-1} X^{2i-p} B(X), \tag{7}$$

where (B, d_B, p) records a predecessor locator, its nonzero discrepancy, and its round index on the doubled schedule. Initially $\Lambda^{(0)} = 1$, with artificial predecessor $(B, d_B, p) = (1, 1, -1)$. If $d_i = 0$, the locator and predecessor remain unchanged. The predecessor record is replaced by $(\Lambda^{(i)}, d_i, 2i)$ exactly when $d_i \neq 0$ and $2i - \ell_i > p - \deg B$, using the locator before the update.

For $d_i \neq 0$, a direct coefficient update $\Lambda'_{r+\kappa} = \Lambda_{r+\kappa} + f B_r$, with $\kappa = 2i - p$, $f = d_i d_B^{-1}$, and $0 \leq r \leq \deg B$, uses an RWD destination index despite the fixed round count. Field arithmetic introduces further indexed reads: for nonzero a, b, log/antilog multiplication computes $ab = \mathcal{A}[(\mathcal{L}[a] + \mathcal{L}[b]) \bmod n]$, where $\mathcal{A}[e] = \alpha^e$ and $\mathcal{L}[x] = \log_\alpha x$. The systematic encoder similarly selects a remainder row $\mathcal{T}[q]$ using a byte q derived from received data. Thus fixing the recurrence's rounds does not remove its indexed coefficient, field, and encoder operations.

WSR's fixed-address implementation scans each indexed domain and masks the selected value; each protected field lookup examines $Q = 2^m$ candidates. The problem is to preserve Equation (7) and the surrounding decoding stages under the trace objective in Equation (5), while eliminating these full-field scans and avoiding RWD addresses.

3.2 Deterministic table-free core

We resolve these indexed operations by algebraic index elimination: field lookups become arithmetic circuits, encoder-row selection becomes linear-basis evaluation, and locator shifts become relations over public coefficient pairs. The construction retains WSR's recurrence and degree rule. For a parity map satisfying Equation (1), the decoder assumes exact, fixed-schedule evaluation of P and field arithmetic. It uses t locator rounds, $(t+1)$-coefficient arrays, and all n Chien points without randomness. Algorithms 1 to 3 specify the $m = 9$, $n = 511$ realization.

Replacing field-table lookups. WSR's log/antilog selections are replaced by polynomial-basis computation. For general m, field multiplication uses m fixed shift-and-reduce rounds modulo a public irreducible polynomial, and inversion uses a fixed addition chain for a^{2^m-2}. A multiword representation uses a public, fixed number of limbs. These arithmetic choices preserve the locator recurrence. General multiplication in $\mathbb{F}_{2^9}$ executes nine shift-and-reduce rounds modulo z^9+z^4+1; squaring uses fixed bit expansion and reduction. Inversion

computes $a^2, a^3, a^{12}, a^{15}, a^{240}, a^{255}, a^{510}$ using, respectively, one square, a multiplication by a, two squares, a multiplication by a^3, four squares, a multiplication by a^{15}, and one square. It maps zero to zero. These operations use neither log/antilog tables nor RWD addresses.

For unsigned machine words x, y and a Boolean bit b, conditional operations use the arithmetic mask $\mathsf{Mask}(b) = 0 - b$ and the selection primitive $\mathsf{Select}(x, y, b) = x \oplus ((x \oplus y) \,\&\, \mathsf{Mask}(b))$. In the algorithms, $[P]$ is the bit representing predicate P, $\texttt{NZ}(x) = [x \neq 0]$, and $\texttt{EQ}(x, y) = [x = y]$. These are fixed-control primitives; Boolean combinations are bitwise, and all field expressions are evaluated before their results are masked or selected. The nonzero-product guards mirror the C masks and suppress only products that are already zero.

Replacing encoder-row selection. To replace the RWD lookup $\mathcal{T}[q]$, use the linearity of the evaluated polynomial encoder's 256-row remainder table. Writing $q = \sum_{b=0}^{7} q_b 2^b$ gives

$$\mathcal{T}[q] = \bigoplus_{b=0}^{7} q_b \mathcal{T}[2^b]. \tag{8}$$

The protected encoder therefore reads the same eight basis rows for every byte. In Algorithm 3, $\texttt{Basis}[b, v]$ is word v of the basis row $\mathcal{T}[2^b]$, and $\texttt{Byte}(a, h)$ packs the eight message bits starting at $8h$ in MSB-first order. For a systematic received word $y = (a, p)$, re-encode a and form the parity discrepancy $\Delta = p \oplus P(a)$. Since $(a, P(a))$ belongs to the BCH parent,

$$S_j(y) = S_j(0, \Delta), \qquad S_{2j}(y) = S_j(y)^2. \tag{9}$$

Syndrome evaluation therefore need only process the parity discrepancy. Every odd syndrome traverses the complete public parity range; a discrepancy bit controls only a masked XOR of a publicly addressed field constant. The Frobenius identity in Equation (9) supplies the even syndromes.

Replacing indexed locator shifts. To replace the indexed destination $r+\kappa$ in Equation (7), express each destination coefficient as a masked sum over public source positions. Let L be any public bound on the intermediate locator degrees; the evaluated bounded-distance decoder takes $L = t$. Store both polynomials in full $(L + 1)$-coefficient arrays. With $\kappa = 2i - p$, coefficient j of the update is computed as

$$\Lambda'_j = \Lambda_j \oplus \bigoplus_{r=0}^{L} [j = r + \kappa]\, d_i d_B^{-1} B_r, \qquad 0 \le j \le L. \tag{10}$$

The inner sum tests every (j, r) pair, so the RWD shift affects values but never an address. When the final locator degree is at most t, exactly t paired rounds establish all $2t$ identities.

Algorithm 1 uses $t+1$ cells per array and stores the predecessor length as $\ell_B = 1 + \deg B$. When $\ell_B \le t$, B_{ℓ_B} is a zero sentinel included in the product loop alongside the live prefix $B_0, \ldots, B_{\ell_B - 1}$. The C implementation allocates $2t + 1$ cells for the predecessor and copy arrays but operates only on their first $t + 1$ cells.

Completing Chien search and correction. The same field operations replace WSR's log-domain Chien lookups. The Chien stage selects and inverts the locator's leading coefficient by scanning the public $t + 1$ coefficient positions. It then evaluates all t nonconstant terms at all n nonzero field points (511 in the specified realization). Each term advances through multiplication by a public α^j value with a fixed reduction schedule, and the specification writes every correction coordinate. A root at α^i identifies the public

```
Fixed-Locator(S[1..2t]):
  Lambda,B,C,U ← four zero arrays of length t+1
  Lambda[0] ← 1; B[0] ← 1
  (d,dB,ell,ellB,p,f) ← (S[1],1,0,1,-1,0)
  for i = 0,...,t-1:
    within ← [ell <= t]; q ← NZ(d) and within
    kappa ← Select(0, 2i-p, q)
    for j = 0,...,t:
      C[j] ← Select(C[j], Lambda[j], q and [j <= ell])
    f ← Select(f, GF-Mul(d, GF-Inv(dB)), q)
    for s = 0,...,t:
      U[s] ← GF-Mul(f,B[s]) and Mask(q and [s <= ellB]) // B[ellB]=0
    for j = 0,...,t:
      v ← Lambda[j]
      for s = 0,...,t:
        v ← v xor (U[s] and Mask([j = s+kappa]))
      Lambda[j] ← v
    ellp ← Select(0, ellB-1+kappa, q)
    u ← q and [ell < ellp]
    ellB ← Select(ellB, ell+1, u)
    for j = 0,...,t:
      B[j] ← Select(B[j], C[j], u and [j <= ell])
    ell ← Select(ell,ellp,u)
    dB ← Select(dB,d,u); p ← Select(p,2i,u)
    next ← Select(1,2i+3,[i < t-1])              // public
    active ← [i < t-1] and within; d ← Select(d,S[next],active)
    for j = 1,...,t:
      idx ← (2i+3-j) if j <= 2i+2, else 1  // public
      use ← active and [j <= ell] and NZ(Lambda[j]) and NZ(S[idx])
      d ← d xor (GF-Mul(Lambda[j],S[idx]) and Mask(use))
  return (Lambda,ell)
```

Algorithm 1: Paired locator construction. Every loop bound and array address shown is public.

correction coordinate $\chi(i) = \pi^{-1}(-i \bmod n)$; for the serialized map in Equation (4), $\chi(i) = i - 1$. The deterministic API returns the systematic candidate together with a status: success requires a locator degree between zero and t, a matching root count, and valid public format. The optional position-randomizing wrapper (Section 3.6) additionally masks its output to zero on failure. The two format predicates in Algorithm 3 enforce the unused-systematic-position and reserved-bit rules of Section 2.

The deterministic C Chien routine applies systematic corrections directly; the specification and position-randomizing wrapper materialize the full correction vector. These realizations are algebraically equivalent and use public iteration bounds and addresses.

```
Complete-Chien(Lambda[0..t], ell):
  degree_ok ← [0 <= ell] and [ell <= t]; lead ← 0
  for j = 0,...,t:
    lead ← lead xor (Lambda[j] and Mask([j = ell]))
  g ← GF-Inv(lead)
  c ← GF-Mul(Lambda[0],g) and Mask(NZ(Lambda[0]) and NZ(lead))
  for j = 1,...,t:
    below ← [j < ell] and degree_ok; leading ← [j = ell] and degree_ok
    T[j] ← GF-Mul(Lambda[j],g) and Mask(below and NZ(Lambda[j]))
    T[j] ← GF-Mul(T[j],alpha^j) xor (alpha^j and Mask(leading))
  e[0..510] ← 0; count ← 0; done ← 0
  for i = 1,...,511:
    v ← c
    for j = 1,...,t: v ← v xor T[j]
    root ← EQ(v,0) and [done = 0] and degree_ok
    e[i-1] ← root                          // fixed public write
    count ← count + root
    done ← done or (root and [count = ell])
    for j = 1,...,t: T[j] ← GF-Mul(T[j],alpha^j)
  return (e,count,[count = ell] and degree_ok)
```

Algorithm 2: Complete Chien search. Every loop bound and array address shown is public.

```
CW-Decode(y = (a,parity)):
  R ← zero r-bit encoder remainder
  for each public message-byte position h:
    q ← MSB8(R) xor Byte(a,h); row ← zero r-bit row
    for b = 0,...,7:
      for each public parity word v:
        row[v] ← row[v] xor (Basis[b,v] and Mask(Bit(q,b)))
    R ← (R shifted left by 8 within r bits) xor row
  Delta ← parity xor R; S[1..2t] ← 0
  for h = 0,...,r-1:
    for j = 0,...,t-1:
      idx ← (2j+1)*pi(k+h) mod 511       // public index
      S[2j+1] ← S[2j+1] xor (alpha^idx and Mask(Delta[h]))
  for j = 1,...,t: S[2j] ← GF-Square(S[j])
  (Lambda,ell) ← Fixed-Locator(S)
  (e,w,roots_ok) ← Complete-Chien(Lambda,ell)
  for q = 0,...,k-1: candidate[q] ← a[q] xor e[q]
  format_ok ← Unused-Systematic-Zero(candidate)
              and Reserved-Storage-Zero(y)
  ok ← roots_ok and [w = ell] and format_ok
  for each public output bit q: out[q] ← candidate[q]
  erase temporary arrays and intermediate scalar state
  return (out,ok)
```

Algorithm 3: $m = 9$, $n = 511$ specification of the deterministic table-free decoding core. Every loop bound and array address shown is public. The inner boxes mark basis encoding and the field/locator/Chien/correction realizations that differ from WSR; its paired recurrence and full Chien traversal are inherited.

3.3 Correctness and trace regularity

Lemma 1 (Systematic syndrome reduction)**.** *Let P and $\mathcal{C}_P$ satisfy Equation (1). The parity discrepancy $\Delta = p \oplus P(a)$ has exactly the BCH syndromes of $y = (a, p)$, and the even syndromes follow by squaring.*

Proof. Adding the parent-code word $(a, P(a)) \in \mathcal{C}_P \subseteq \mathcal{B}$ to y gives $(0, \Delta)$ without changing a BCH syndrome. For binary coordinates, Frobenius squaring sends $\sum_q y_q \alpha^{j\pi(q)}$ to $\sum_q y_q \alpha^{2j\pi(q)}$. □

Lemma 2 (Algebraic equivalence of index elimination)**.** *Let $S_1, \ldots, S_{2t}$ be binary BCH power-sum syndromes. Consider the paired recurrence of Equation (7), initialized with $(\Lambda, B, d_B, p) = (1, 1, 1, -1)$ and replacing the predecessor at round i exactly when $d_i \neq 0$ and $2i - \deg \Lambda > p - \deg B$. When $d_i = 0$, the locator and predecessor are unchanged. For any public L bounding its intermediate degrees, the coefficient-pair relation in Equation (10) produces the same locator, predecessor, and degree after every round as the shifted polynomial update. After t rounds this locator has constant term one and least degree among constant-one polynomials satisfying the first $2t$ Newton identities. In particular, for an error set E of weight $w \leq t$, taking $L = t$ produces (Λ_E, w) with a nonzero leading coefficient.*

Proof. Equation (10) expands the shifted polynomial update coefficient by coefficient, so it preserves every state when L bounds the intermediate degrees. It remains to establish the paired recurrence's degree and correctness properties.

Extend the syndromes as binary power sums. Let D denote formal differentiation, put $G(X) = \sum_{h \geq 1} S_h X^{h-1}$, and define the Newton residual $R_C = CG + DC$. Its coefficient at X^{h-1} is the residual of identity h. Binary conjugacy gives $DG = G^2$; since $D^2 C = 0$ in characteristic two, $DR_C = R_C G$. If $R_C = O(X^{2i+1})$, comparison of X^{2i} coefficients in this identity forces the X^{2i+1} coefficient of R_C to vanish. Thus satisfying odd identity $2i + 1$ also satisfies identity $2i + 2$.

At the start of round i, we maintain that $\Lambda(0) = 1$, that Λ has least degree $\ell \le 2i$ among constant-one solutions of the first $2i$ identities, and that $R_\Lambda = d_i X^{2i} + O(X^{2i+1})$. The predecessor is either the artificial $(B, d_B, p) = (1, 1, -1)$ or a pre-update locator from round $s < i$ with $p = 2s$, $R_B = d_B X^p + O(X^{p+1})$, and $d_B \neq 0$. In both cases $p - \deg B = \ell - 1$ and $\deg B \le \ell$. The initialized state satisfies these invariants.

If $d_i = 0$, the residual identity supplies the next even identity; any lower-degree solution would also solve the previous prefix, so minimality persists. Otherwise put $\kappa = 2i - p > 0$. For a real predecessor, κ is even, so $R_{X^\kappa B} = X^\kappa R_B = d_B X^{2i} + O(X^{2i+1})$. For the artificial predecessor, $\kappa = 2i + 1$ and $R_{X^\kappa} = X^{2i} + O(X^{2i+1})$, giving the same formula with $d_B = 1$. The update therefore cancels the odd discrepancy, establishes the next two identities, and preserves the constant term one.

The shifted term has degree $\ell^\star = \deg B + 2i - p = 2i + 1 - \ell$. Since $\ell + \ell^\star$ is odd, the leading terms cannot cancel:

$$\deg \Lambda' = \max(\ell, \ell^\star). \tag{11}$$

Any constant-one solution C of the enlarged identity prefix satisfies $D(C\Lambda) = CR_\Lambda + \Lambda R_C$, whose coefficient at X^{2i} is $d_i \neq 0$. The bound $\deg D(C\Lambda) \le \deg C + \ell - 1$ gives $\deg C \ge 2i + 1 - \ell$; previous minimality gives $\deg C \ge \ell$. Equation (11) attains both bounds. The predecessor rule replaces the record exactly when $\ell^\star > \ell$; then its new score is $2i - \ell = \ell^\star - 1$ and its residual is the pre-update R_Λ. Otherwise the record is retained. Hence all invariants persist, degrees are nondecreasing, and the degree after round i is at most $2i + 1$.

For $w = 0$, minimality gives $\Lambda = 1$. For $1 \le w \le t$, logarithmic differentiation of Λ_E gives the error power sums, so $R_{\Lambda_E} = O(X^{2t})$ and minimality gives $\deg \Lambda \le w$. The identity $D(\Lambda\Lambda_E) = \Lambda_E R_\Lambda + \Lambda R_{\Lambda_E} = O(X^{2t})$ then vanishes identically because its degree is at most $2t - 1$. The error-location factors are distinct, so each root ξ of Λ_E satisfies $(D\Lambda_E)(\xi) \neq 0$. Evaluating the zero derivative gives $\Lambda(\xi)(D\Lambda_E)(\xi) = 0$, hence Λ_E divides Λ. The degree bound and constant terms give $\Lambda = \Lambda_E$. Nondecreasing degrees now imply that $L = t$ suffices throughout. □

Lemma 3 (Faithful bounded-array realization)**.** *If WSR's paired recurrence finishes with degree at most t, the* Fixed-Locator *routine in Algorithm 1 realizes the recurrence of Lemma 2 exactly; in particular, neither its bounded arrays, predecessor sentinel, nor dummy syndrome index changes the result.*

Proof. Locator degrees are nondecreasing by Equation (11); hence the terminal degree bound implies that no intermediate degree exceeds t, and `within` is always true. The initialized arrays and each update keep $\Lambda_j = 0$ for $j > \ell$. They also maintain $B_j = 0$ for $\ell_B \le j \le t$: on predecessor replacement, B receives the pre-update locator through degree ℓ, while its retained tail is zero because predecessor degrees cannot exceed the current locator degree. Thus $\ell_B = 1 + \deg B$, the extra product selected at $s = \ell_B$ is the documented zero sentinel, and the all-pairs loop implements Equation (10) without truncation.

After round i, Lemma 2 gives $\ell \le 2i + 1$. Consequently every term assigned the dummy syndrome index has $j > 2i + 2 > \ell$ and is masked out; all remaining terms use the true index $2i + 3 - j$. The nonzero guards suppress only zero products, so the next discrepancy is exact. The copied predecessor, degree update, and discrepancy are therefore identical to the algebraic state in every round. □

Lemma 4 (Chien search and root counting)**.** *On input (Λ_E, w) for $|E| = w \le t$, the complete Chien stage returns the indicator vector of E, root count w, and a true validity bit.*

Proof. Write $\Lambda_E(X) = \sum_{j=0}^{w} \lambda_j X^j$. Lemma 2 gives $\lambda_w \neq 0$, so normalization by λ_w is defined and preserves all roots. At Chien iteration i, the jth maintained term is $(\lambda_j/\lambda_w)\alpha^{ij}$;

initialization gives this relation at $i = 1$, and multiplication by public α^j preserves it from one iteration to the next. The tested value is therefore $\lambda_w^{-1}\Lambda_E(\alpha^i)$.

The points $\alpha, \ldots, \alpha^n$ enumerate $\mathbb{F}_{2^m}^*$ exactly once. In the generic construction, iteration i writes the public coordinate $\chi(i) = \pi^{-1}(-i \bmod n)$, which belongs to E exactly when α^i is a root of Λ_E. For the evaluated coordinate map in Equation (4), $\chi(i) = i - 1$. A nonzero polynomial of degree w has at most w roots, and Λ_E already has the w prescribed roots. Thus no valid root is suppressed and no extra correction is introduced. The returned vector, count, and validity bit follow; the clean case $w = 0$ is immediate from $\Lambda_E = 1$. □

Theorem 1 (Bounded-distance correctness)**.** *Let $\mathcal{B}$ be a primitive narrow-sense binary BCH code of length $n = 2^m - 1$, and let t be a positive decoding radius at most its designed radius. Let P, $\mathcal{C}_P$, and π satisfy the parameterized construction above, and instantiate it with exact field arithmetic and exact evaluation of P. If $c = (a, P(a)) \in \mathcal{C}_P$ and $y = c + e$ with $\mathrm{wt}(e) \le t$, its internal Chien correction vector is e and its systematic candidate is a. If the public-format predicate accepts c and every checked storage position outside the n BCH coordinates retains its valid public value, the returned validity bit is true. The deterministic API returns the candidate; an output mask under this bit also releases a under the format premise.*

Proof. Exact evaluation of P and Lemma 1 supply all $2t$ syndromes of e. The parent BCH distance is at least $2t + 1$, so the correction is unique. By Lemma 2, the algebraic recurrence returns $(\Lambda_E, |E|)$ with terminal degree $|E| \le t$. This satisfies the hypothesis of Lemma 3, so the bounded-array routine returns the same pair. By Lemma 4, Chien search returns exactly e and the correct root count. Correcting the systematic positions restores a. Under the additional format assumption, every acceptance condition passes, so the validity bit is true and masking under it preserves a. □

For $\mathrm{wt}(e) > t$ the decoder makes no correctness guarantee. It nevertheless executes every public loop and completes candidate and status production, so it does not present a stage-presence oracle. Success means that degree, root count, and public format checks passed; it does not certify distance at most t from $\mathcal{C}_P$. For example, the received polynomial g_{29} lies entirely in the parity positions of $\mathcal{C}_{384}$ and has zero BCH syndromes, so it is accepted with zero candidate although it is outside $\mathcal{C}_{384}$. The analogous example for $\mathcal{C}_{320}$ is g_{45}. KEM-level deterministic re-encryption, constant-time comparison, and implicit rejection remain mandatory; the candidate and decoder status must stay internal to that boundary.

Corollary 1 (Correctness of the two code instantiations)**.** *The decoder in Algorithm 3, instantiated with the displayed $\mathbb{F}_{2^9}$ arithmetic, eight-basis polynomial encoders, serialized coordinate map, and public format rules, satisfies Theorem 1 for $\mathcal{C}_{384}$ and $\mathcal{C}_{320}$. In particular, this includes the noncyclic $\mathcal{C}_{320}$.*

Proof. Equations (2) and (3) use monic polynomials $H(x)$ of degree $r = 511 - k$. With the MSB-first message polynomial $a(x) = \sum_{q=0}^{k-1} a_q x^{k-1-q}$, the encoder computes $P(a) = x^r a(x) \bmod H(x)$. Thus H divides $x^r a + P(a)$, whose degree is below 511, and its systematic coordinates agree with $\pi(q) = 510 - q$. The systematic identity block has rank k, so the encoder image equals the specified k-dimensional space of multiples of H. Both images lie inside their BCH parents. Equation (8) evaluates these parity maps exactly. The field and coordinate instantiations are those specified in Sections 2 and 3.2. The locator realization is exact by Lemma 3, so the theorem applies; the reserved storage bit lies outside the 511 BCH coordinates and is covered by its format premise. □

Proposition 1 (Source-level trace regularity)**.** *For fixed public parameters and layout, assume that evaluation of P and the selection, field, and cleanup primitives preserve their stated fixed schedules. The parameterized construction, including its concrete instantiation*

in Algorithms 1 to 3, has the same ordered control-flow and memory-address trace for every received word.

Proof. All loop bounds and addresses are public. Received bits, discrepancies, degrees, shifts, roots, and validity predicates control only masks; in particular, locator shifts test every coefficient pair and the Chien stage writes every coordinate. Associate each primitive invocation with its ordered instruction and memory events from Equation (5). By assumption this event sequence depends only on its public arguments. Public loop bounds concatenate the same primitive sequences in the same order, preserving the property through correction, candidate copying, status computation, and cleanup. The direct systematic-correction realization satisfies the same argument with its own fixed sequence. □

3.4 Preserving the source schedule in compiled code

Proposition 1 assumes that primitive traces depend only on public arguments. Applying this guarantee to a compiled decoder additionally requires preservation of the surrounding control flow and memory accesses. C semantics alone guarantee neither property. Apple Clang 21 compiled ordinary C masking expressions into RWD `tbz` branches, even with DIT enabled. This transformation preserves functional outputs but violates the trace objective in Equation (5).

To prevent this transformation in the evaluated binaries, the sources use the compiler-opaque primitive `ct_mask_u32`. It normalizes the Boolean input and computes its negation using target-specific inline assembly; the generic implementation uses a volatile temporary:

```
ct-mask-u32(b):
  uint32_t v ← b and 1; uint32_t n
  AArch64: asm volatile("neg %w0, %w1" : "=r"(n) : "r"(v))
  ARM32:   asm volatile("rsb %0, %1, #0" : "=r"(n) : "r"(v))
  x86:     n ← v; asm volatile("neg %0" : "+r"(n) : : "cc")
  otherwise: volatile uint32_t u ← v; n ← 0U-u
  return n
```

In the audited AArch64 and x86 builds, Boolean expansion emits `neg` followed by logical selection. Preservation must still be checked for each binary and across its protected paths; Section 4.2 reports the audit scope. A preserved schedule does not imply operand-independent instruction timing, which Section 4.3 evaluates separately.

3.5 Work and implementation footprint

We compare the source-level work of the two realizations by counting protected field-table selections, coefficient-pair tests, and field arithmetic operations.

Proposition 2 (Protected-domain complexity)**.** *For a primitive narrow-sense binary BCH code over $\mathbb{F}_{2^m}$ with correction radius t, the fixed-address WSR syndrome, locator, and Chien stages perform*

$$N_{\mathrm{WSR}}(m,t) = 5t^2 + (Q+6)t + 4 \tag{12}$$

masked field-table selections and therefore examine $QN_{\mathrm{WSR}}(m,t)$ logical field-table candidates. The index-eliminating realization performs no masked selection over the Q-entry field domain; its leading-coefficient selection scans $t+1$ polynomial coefficients.

Proof. Each WSR syndrome square uses two field-table selections, giving $2t$. Each locator round uses two selections for the discrepancy ratio, $2(t+1)$ for predecessor products, and $3t$ for the next discrepancy: $t(5t+4)$ in total. Chien normalization uses three selections, its coefficient initialization uses $t+1$, and the search uses $(Q-1)t$, giving $Qt+4$. Summing gives Equation (12), and each selection scans all Q candidates. Index elimination replaces

field-table selection by arithmetic circuits, and Algorithm 2 scans exactly $t+1$ positions to select the leading coefficient. □

Table 2: Exact protected-domain work by stage for $Q = 2^m$. WSR field candidates count every logical table entry examined; the evaluated $m = 9$ implementation packs seven entries per 64-bit word.

Stage	WSR masked selection	Algebraic index elimination
Encoder	256 remainder-row candidates per data byte and parity word	Eight public basis words per data byte and parity word
Syndromes	$2t$ field selections; $2Qt$ field candidates	Complete parity scan and t fixed squarings
Locator	$t(5t+4)$ field selections; $Qt(5t+4)$ field candidates; $2t(t+1)^2$ coefficient candidates	t inversions; $2t^2+2t$ multiplications; $t(t+1)^2$ coefficient-pair tests
Chien	$Qt+4$ field selections; $Q(Qt+4)$ field candidates	One inversion; $t+1$ multiplications; Qt fixed-α^j updates; $(Q-1)t$ accumulations
Total field candidates	$Q\left(5t^2+(Q+6)t+4\right)$	None

For index elimination, the locator and Chien counts in Table 2 follow from the public loops in Algorithms 1 and 2: each of the t locator rounds visits $(t+1)^2$ coefficient pairs, while the complete Chien schedule initializes t terms and advances them across all $Q-1$ nonzero field points.

The operations counted in Table 2 have different costs. For the three stages in Proposition 2, we use a word-RAM model with w-bit words and $m \le w$; WSR scans compare $b = \lfloor w/m \rfloor$ packed field entries per word in parallel. Let $M(m)$ and $I(m)$ denote the costs of general multiplication and inversion, and let A_j be the cost of multiplication by the public constant α^j in the chosen realization.

The combined work of these three stages is

$$W_{\mathrm{WSR}} = \Theta\left(\left\lceil \frac{Q}{b} \right\rceil N_{\mathrm{WSR}}(m,t) + t^3\right), \tag{13}$$

$$W_{\mathrm{AIE}} = \Theta\left(t^3 + t^2 M(m) + tI(m) + Q\sum_{j=1}^{t} A_j\right). \tag{14}$$

For generic m-round bit-serial multiplication, $M(m), A_j = \Theta(m)$; then the Chien term dominates the coefficient-pair term when $t^2 = o(Qm)$. The measured scalar specialization instead uses a public shift and fixed reduction via $z^9 = z^4 + 1$, so A_j depends on the exponent and the reduction circuit; operation counts alone do not determine runtime.

At $t = 22$ and $Q = 512$, WSR's three stages examine 7,075,840 logical field-table candidates (1,022,680 reads of packed 64-bit table words), while index elimination uses 11,638 locator coefficient-pair tests and 11,264 fixed-α^j Chien updates.

Carrying the shifted predecessor $X^{2i-p}B/d_B$ instead of B permits public shifts with $O(t)$ coefficient operations per round. We retain WSR's predecessor state for a direct equivalence argument and matched implementation comparison; the all-pairs bound characterizes this realization without asserting optimality.

Table 3 reports the combined footprint of the encoder and deterministic checked decoder for each code. The AArch64 VM builds use GCC 11.4 with `-march=armv8.4-a`, at both `-O3` and `-Os`, with separate function and data sections. Resource accounting includes only the encode/decode call graphs and the table entries they use. Estimated peak stack is the maximum sum of `-fstack-usage` frames along a retained call path, excluding library callees. The "protected tables" column counts bytes in field and encoder tables with received-word-dependent selections, excluding publicly indexed constants.

On AArch64, the `-O3` table-free `.rodata` contains only the publicly indexed 1,024-byte power table; GCC folds the eight encoder basis rows into code. With `-Os`, the basis rows

Table 3: Combined footprint of the encoder and deterministic checked decoder on the AArch64 VM, in bytes. In each paired entry, the left value uses `-O3` and the right value uses `-Os`. Protected-table sizes are identical at both levels. The WSR baselines adapt the released masked-table code to matching parameters. Each object has eight bytes of writable relocation state and no `.bss`.

Code	implementation	`.text`	`.rodata`	est. peak stack	protected tables
$\mathcal{C}_{384}$	WSR baseline	8,488/4,020	6,304/6,304	912/848	5,280
	Table-free core	11,076/3,696	1,024/1,152	1,376/928	0
$\mathcal{C}_{320}$	WSR baseline	5,760/4,028	8,352/8,352	1,136/1,088	7,328
	Table-free core	9,824/3,844	1,024/1,216	1,504/1,216	0

occupy 128/192 additional bytes for $\mathcal{C}_{384}/\mathcal{C}_{320}$. Relative to `-O3`, reduced unrolling and inlining lower the combined code and read-only footprint by 59.9%/53.4%, respectively. The WSR baselines retain the power table and require 1,184 bytes of packed log/power tables plus 4,096/6,144 bytes of encoder remainder rows. Table elimination costs additional code under `-O3`; under `-Os`, the table-free cores use less code than these WSR baselines. Their stack footprints remain larger at both optimization levels.

On the Intel i7-13700 (GCC 12.2, `-march=native`), switching from `-O3` to `-Os` reduces the table-free code and read-only footprint by 65.6%/60.0% for $\mathcal{C}_{384}/\mathcal{C}_{320}$, respectively. The resulting 4,536/4,878-byte footprints save 5,441/7,162 bytes relative to the matched `-Os` WSR baselines, with 112/144 additional bytes of estimated peak stack.

For Cortex-M4, Arm GNU 13.2.Rel1 cross-builds of the table-free cores save 2,600/3,492 bytes of Flash with `-O3` and 5,434/7,310 bytes with `-Os` for $\mathcal{C}_{384}/\mathcal{C}_{320}$, respectively, relative to the matched fixed-address WSR baselines. Additional peak stack is 444/324 bytes with `-O3` and 132/164 bytes with `-Os`.

3.6 Optional position-randomizing wrapper

A fixed execution schedule does not hide the values processed by the decoder. The optional wrapper combines a random codeword mask with a random cyclic shift of the error positions to vary operands across calls.

For $y = c + e$ with $c \in \mathcal{C}_P$, choose $u \in \mathbb{F}_2^k$ uniformly and form $y_b = y + b$, where $b = \mathsf{Enc}(u) \in \mathcal{C}_P$. Linearity gives $y_b = (c + b) + e$ and $S_j(y_b) = S_j(e)$: masking changes the data representation but preserves the error syndromes.

Independently choose $\rho \in \mathbb{Z}_{511}$ uniformly and set $\gamma = \alpha^{-\rho}$. Under Equation (4), shifting an error from position q to $(q + \rho) \bmod 511$ multiplies its contribution to syndrome j by γ^j. We therefore compute $S'_j = \gamma^j S_j(y_b)$ for $1 \le j \le 2t$, obtaining the BCH syndromes of the shifted error vector. Locator construction and Chien search operate on these transformed syndromes.

Rotation preserves error weight, so for $\mathrm{wt}(e) \le t$ the locator and Chien stages recover the shifted error vector by Lemmas 2 and 4. A nine-stage network restores the original coordinates. With $d = (-\rho) \bmod 511$, stage $s = 0, \ldots, 8$ selects the cyclic shift by 2^s when bit s of d is one and the unchanged vector otherwise. Correcting the masked word and removing b gives $y_b + e + b = c$. Since only the error coordinates are shifted, $\mathcal{C}_P$ need not be cyclic. Under the format premises of Theorem 1, the root-count and format checks pass and the original payload is returned. The wrapper masks its output to zero on failure; beyond radius t, miscorrection remains possible and KEM-level re-encryption and implicit rejection are still required.

Within the protected call, field exponentiation uses fixed rounds, and each network selection reads both public source coordinates before applying a mask. Its loop bounds

and addresses are thus independent of y and $\omega = (u, \rho)$, satisfying the joint trace objective in Section 2.2 under the primitive assumptions of Proposition 1. This entire path is the *complete wrapper*; the deterministic core requires neither randomization step.

If a fresh uniform shift is not observed elsewhere, error patterns differing only by rotation induce the same distribution of syndrome-derived locator/Chien operands: adding a fixed offset to ρ preserves uniformity. This restricted view excludes syndrome computation and transformation, inverse rotation, and correction. Error weight, cyclic geometry, and zero syndromes remain invariant; no power, EM, or SASCA resistance follows from this property.

4 Implementation evaluation

The evaluation covers correctness, ordered instruction-and-address traces, timing, and performance for both codes. Decoder comparisons use WSR's fixed-address masked-table strategy, including its two published shortened codes (Section 4.4); the ANSA-IBKEM integration uses $\mathcal{C}_{384}$.

4.1 Evaluation design

Table 4 gives the six platforms and decoding radii. Primary RWD timing screens run on the physical M5, Cortex-A76, i7-13700, and bare-metal STM32F407. We include AArch64 and x86-64 VMs to investigate decoder behavior under virtualization. The AArch64 VM supplies controlled instruction-and-address traces and resource measurements, while the Broadwell-EP VM provides supporting timing evidence on an older compiler.

Table 4: Evaluation platforms. The x86-64 leakage screens use invariant-TSC reference ticks. Latencies in Table 8 use `CLOCK_MONOTONIC_RAW` on the i7 and M5 for both codes, and on the Raspberry Pi and both VMs for $\mathcal{C}_{320}$.

Target	processor	operating system	compiler	timer	(t_{384}, t_{320})
AArch64 VM	Parallels ARM, 8 vCPU	Linux 5.15	GCC 11.4	CNTVCT_EL0	(14, 22)
Apple M5	Apple M5	macOS 26.5.1	Apple Clang 21	CNTVCT_EL0	(14, 22)
Raspberry Pi 5	Cortex-A76 r4p1	Linux 6.18	GCC 14.2	CNTVCT_EL0	(14, 22)
Intel i7-13700	Raptor Lake	Linux 6.8	GCC 12.2	RDTSC / monotonic raw	(14, 22)
Broadwell-EP VM	Xeon E5-2620 v4	Linux 3.10	GCC 4.8.5	LFENCE/RDTSC	(14, 22)
STM32F407	Cortex-M4 / ARMv7E-M	bare metal	Arm GNU 13.2.Rel1	DWT cycle counter	(14, 22)

Hosted timing and performance builds use `-O3`. The M5 and AArch64 VM trace/audit builds target `armv8.4-a`; the x86-64 and Cortex-A76 builds use `-march=native`. The STM32F407 scalar build uses `-O3 -mthumb -mcpu=cortex-m4 -mfloat-abi=hard -mfpu=fpv4-sp-d16`. Each M5 process sets and verifies the selected PSTATE.DIT value before warm-up; this setup is excluded from measurement. M5 and Cortex-A76 counters run at 1 GHz and 54 MHz, respectively.

The hosted timing campaign compares clean and one-error words, errors at coordinates 0 and 510, and weights t and $t + 1$, under fixed and fresh wrapper coins. The *prepared* wrapper receives its coins from the caller; the *fresh* wrapper includes their generation. Each repetition uses balanced randomized class order and 100,000 observations per class. Six aggregate contrasts and three contrasts across eight stage metrics give 30 comparisons per repetition, or 90 over three independent seeds.

The STM32F407 DWT screens cover eleven input classes, the core and prepared wrapper, and three repetitions of 100,000 observations per class, yielding 6.6 million observations per code. Input construction is equalized, and fixed front-end padding lies outside the timed interval.

We summarize each comparison by the class means and standard deviations, an unadjusted pointwise 95% interval for $\mu_0 - \mu_1$, and, when defined, the signed Welch t

statistic and Hedges' g. Because Table 6 selects contrasts by maximum $|t|$, its intervals are descriptive and do not have simultaneous or post-selection 95% coverage.

We use $|t| > 4.5$ as a Test Vector Leakage Assessment (TVLA)-style mean-alert threshold [GJJR11, RBV17]. With 100,000 balanced observations per class, a normal equal-variance approximation gives 80% power at a standardized mean difference of approximately 0.024 at that alert threshold. For hosted primary scalar comparisons, we additionally test dispersion with Brown–Forsythe and distributions with Epps–Singleton. Bonferroni correction uses familywise level 0.05 separately for each test type, excluding undefined statistics. The $\mathcal{C}_{384}$ family combines M5 with DIT enabled, Cortex-A76, and both x86 targets, with 360 Brown–Forsythe and 337 applicable Epps–Singleton tests. The separate $\mathcal{C}_{320}$ M5 DIT-on, Cortex-A76, and combined-x86 families have respective test counts of 90/72, 90/76, and 180/180 in the same order.

4.2 Correctness, trace, and binary validation

Theorem 1 establishes algorithmic correctness. The C tests cover all 511 single-error positions and rotation distances, random patterns at weights $0, \ldots, t+1$, reserved storage, and payload boundaries. The scalar implementation and the variant using the Arm PMULL polynomial-multiply instruction agree with the WSR baselines at matching code parameters on the tested cases.

Valgrind Lackey traces on the AArch64 VM cover each complete wrapper for 42 received words paired with three fixed coin sets. All 126 traces per code are identical. The $\mathcal{C}_{384}$ traces contain 241,571 instructions, 12,408 loads, and 5,525 stores; the corresponding $\mathcal{C}_{320}$ counts are 585,566, 33,083, and 12,812. This supports the trace objective for these binaries and tested pairs, not its universal validity.

With the mask primitive of Section 3.4, manual audits of the five hosted $\mathcal{C}_{384}$ wrapper binaries and the $\mathcal{C}_{320}$ M5, AArch64-VM, and x86 binaries, including their called helpers, found no RWD branch or address. The audits cover locator and Chien loop bounds, inverse rotation, and cleanup. The stage harnesses are separate diagnostic binaries; timing them does not extend the aggregate binary audit.

4.3 Operand timing and DIT

Fixed control flow and addresses do not by themselves guarantee input-independent execution time. For example, clean BCH words have zero syndromes and many zero intermediates in locator construction and Chien search; errors change these operands within the same execution schedule. If instruction latency or throughput depends on operand values, these differences can affect runtime.

We assess timing leakage in the table-free decoder through two complementary summaries of the same measurements. Table 5 reports the largest $|t|$ across aggregate and stage comparisons to identify mean-timing alerts for each code and target. Table 6 gives signed differences, effect sizes, and statistics across repetitions for the worst contrasts, showing their magnitude and consistency.

On M5, both summaries include measurements with DIT enabled and disabled. DIT specifies operand-independent timing for covered Arm instructions [Arm]. Among the reported configurations, only the DIT-disabled M5 screens exceed the mean-alert threshold of 4.5.

The contrasts in Table 6 are selected from 1,764 comparisons. Here, *1* denotes an error at coordinate zero, and 0/510 compares errors at the two endpoint coordinates.

For both codes, the DIT-disabled M5 clean/one-error contrasts have consistent signs and $|t| > 4.5$ in all three repetitions. In the tested M5 scalar configurations, the mean screens pass with DIT enabled and fail with DIT disabled. The measurements do not identify the instruction or microarchitectural mechanism responsible for the differences.

Table 5: Scalar backend: maximum $|t|$ over three repetitions of 100,000 observations per class. Stage is the maximum over the individually measured stages. "Exact" denotes equal constant samples, for which Welch's statistic is undefined.

	$\mathcal{C}_{384}$		$\mathcal{C}_{320}$	
Target and mode	aggregate	stage	aggregate	stage
Apple M5, DIT enabled	1.657	2.235	1.869	2.546
Apple M5, DIT disabled	38.626	61.873	47.232	60.422
Cortex-A76	2.355	2.548	1.978	3.499
STM32F407 DWT	exact	exact	exact	exact
Intel i7-13700	2.152	2.184	2.255	2.552
Broadwell-EP VM	2.417	1.779	1.695	2.543

Table 6: Worst signed contrast per code and target mode. Each STM32F407 contrast has equal constant samples in both classes ($\Delta = 0$), so one representative per code is shown. Means and $\Delta = \mu_0 - \mu_1$ use the platform counter (DWT cycles on STM32F407); M5 ticks are nanoseconds. Means are rounded; intervals are descriptive. The $t_{1:3}$ values are listed in repetition order; dashes indicate undefined t and g for equal constant samples.

Target/mode	Code	contrast	μ_0/μ_1	Δ [95%]	$t_{1:3}$	g
M5, DIT on	$\mathcal{C}_{384}$	total, 14/15	9745.92/9752.20	-6.28 $[-11.78, -0.77]$	$0.77/-0.27/-2.23$	-0.010
	$\mathcal{C}_{320}$	synd., clean/1	3802.15/3805.73	-3.58 $[-6.33, -0.82]$	$-2.55/-0.64/0.27$	-0.011
M5, DIT off	$\mathcal{C}_{384}$	corr., clean/1	508.31/534.02	-25.71 $[-26.52, -24.90]$	$-57.21/-61.87$ -21.26	-0.277
	$\mathcal{C}_{320}$	locator, clean/1	7069.98/7208.96	-138.98 $[-143.49, -134.47]$	$-53.29/-60.00$ -60.42	-0.270
Cortex-A76	$\mathcal{C}_{384}$	synd., 14/15	430.304/430.402	-0.098 $[-0.174, -0.023]$	$-2.55/0.34/-0.27$	-0.011
	$\mathcal{C}_{320}$	synd., 22/23	1006.571/1006.713	-0.143 $[-0.223, -0.063]$	$0.71/-0.32/-3.50$	-0.016
i7-13700	$\mathcal{C}_{384}$	mask, clean/1	892.778/893.575	-0.797 $[-1.511, -0.082]$	$-2.18/-0.35/-0.38$	-0.010
	$\mathcal{C}_{320}$	Chien, 0/510	21601.08/21607.83	-6.75 $[-11.94, -1.57]$	$-2.55/-1.40/0.60$	-0.011
Broadwell VM	$\mathcal{C}_{384}$	fresh, clean/1	94370.20/94397.76	-27.56 $[-49.90, -5.21]$	$-0.84/-2.42/-0.05$	-0.011
	$\mathcal{C}_{320}$	Chien, 0/510	74761.56/74728.21	33.35 $[7.64, 59.06]$	$2.54/-0.29/-0.49$	0.011
STM32F407	$\mathcal{C}_{384}$	prep., clean/1	343939/343939	0 $[0, 0]$	–	–
	$\mathcal{C}_{320}$	core, clean/1	833159/833159	0 $[0, 0]$	–	–

The DIT-enabled result applies to the audited binaries with PSTATE.DIT set and verified in the thread executing the decoder. DIT does not repair RWD branches or addresses or prevent physical data leakage; support for FEAT_DIT alone does not provide a portable guarantee for the decoder.

The other tested physical targets pass the mean screens without DIT, and the Broadwell-EP VM also produces no mean alert. On STM32F407, all 324 $\mathcal{C}_{384}$ and 540 $\mathcal{C}_{320}$ comparisons are exact cycle-count equalities at the measured DWT boundary. None of the hosted primary scalar families has a corrected Epps–Singleton rejection. Within these families, the only corrected Brown–Forsythe rejection is the $\mathcal{C}_{320}$ Cortex-A76 fixed-coin syndrome contrast at weights 22/23 in repetition 3 ($p_{\text{raw}} = 0.000393$); it does not recur in the other two repetitions.

4.4 Matched comparison with WSR

Table 7 compares the decoders for our code instances and WSR's two published shortened codes: $(509, 392)$ with radius 13 and $(507, 264)$ with radius 29 [WSR20, WSR19]. Their BCH parents have $(n, k, \delta) = (511, 394, 27)$ and $(511, 268, 59)$; the two implementations fix and omit two and four systematic coordinates, respectively. All decoders scan the 511 parent coordinates. For the shortened codes, the table-free realizations preserve the shortening maps, parity packing, and correction radii while replacing field arithmetic,

locator shifts, encoding, Chien search, and boundary handling.

The released WSR correction loop passes `MAX_ERROR+1` as the scan length to `data_read` and `data_write`, omitting the tail of the systematic byte array. Single-error decoding first fails at systematic coordinates 112 and 240 for radii 13 and 29, respectively, under both `-O0` and `-O3`. The native-corrected variant replaces these two length arguments with `DATA_LEN`, so each masked read and write scans the entire systematic field with a public, fixed bound. The compiler-preserved version additionally protects received-data-dependent encoder accesses and preserves the masked scans. We call these corrected, compiler-preserved adaptations of WSR's released masked-table code at matching parameters the *fixed-address WSR baselines*. All reported speedups use these baselines and are specific to the evaluated implementations and compiler settings. This performance comparison does not assume operand-independent timing for WSR or rank physical side-channel resistance.

The hosted comparison uses a physical Intel i7-13700 and GCC 12.2 with `-O3 -march=native`. Each value is the median of three run means, with 5,000 clean and 5,000 one-error decodings per implementation in each run. Implementation order varies between runs, while the timed boundary is fixed. On Cortex-M4, run means use 1,000 calls per table-free implementation and 20 per WSR implementation; $\mathcal{C}_{320}$ values are medians of three runs.

Table 7: Matched decoder cost. Intel i7-13700 rows are in microseconds; STM32F407 rows are in cycles. Native-corrected WSR repairs only the scan bounds. Speedup is the cost of the fixed-address WSR baseline divided by the complete-wrapper cost.

Code	case	WSR native-corrected	WSR baseline	table-free core	complete wrapper	speedup
Intel i7-13700 (microseconds)						
$(509, 392)$, $t = 13$	clean	840.45	908.10	11.03	12.86	70.60
	one error	840.27	908.98	11.03	12.85	70.72
$(507, 264)$, $t = 29$	clean	2,165.75	2,283.81	63.93	66.45	34.37
	one error	2,167.44	2,285.03	63.65	66.34	34.44
$\mathcal{C}_{320}$	clean	–	1,649.39	28.57	30.88	53.42
	one error	–	1,648.35	28.56	30.89	53.36
STM32F407 at 168 MHz (cycles; complete wrapper includes randomness generation)						
$\mathcal{C}_{384}$	mean	–	62,525,462	317,447	348,634	179.34
$\mathcal{C}_{320}$	mean	–	104,916,001	833,291	889,602	117.94

For the published WSR codes, protecting `mod8_tab` and preserving the masked scans adds 5.4–8.2% to native-corrected WSR latency. Our complete wrappers are 34.37–70.72× faster than the fixed-address baseline on the i7. At radii 13 and 29, the deterministic cores are 82.3–82.4× and 35.7–35.9× faster, respectively. The work counts in Table 2 show the algorithmic difference: the WSR instances examine 3.88 and 9.85 million logical field-table candidates per decode, whereas the table-free cores use fixed-round register arithmetic and small all-pairs locator updates. The measurements compare complete realizations and do not apportion the gain among individual transformations.

On Cortex-M4, the $\mathcal{C}_{384}$ and $\mathcal{C}_{320}$ fresh wrappers are respectively 179.34× and 117.94× faster than the fixed-address WSR baselines at matching code parameters.

4.5 Protection cost and system impact

To distinguish decoder overhead from system-level cost, we first compare the complete wrapper with a conventional clean-word fast path. The fast path returns on zero syndromes and is therefore timing-unsafe. The complete wrapper adds constant work, fresh randomness, random codeword masking, coordinate randomization, final selection, and cleanup. Hosted latencies are medians of three run-level means. On STM32F407, the $\mathcal{C}_{384}$ value averages 1,000 calls; $\mathcal{C}_{320}$ values are medians of three 100,000-call run means.

Table 8: Clean-word latency, in nanoseconds. The fast path is timing-unsafe; the complete wrapper includes fresh randomness, codeword masking, and coordinate randomization. M5 uses DIT.

	$\mathcal{C}_{384}$		$\mathcal{C}_{320}$	
Platform	fast path	wrapper	fast path	wrapper
AArch64 VM	211.802	10,074.711	352.126	23,196.339
Apple M5	295.340	9,951.888	272.432	19,894.523
Raspberry Pi 5	991.602	37,185.966	1,343.854	90,614.306
Intel i7-13700	277.239	13,938.319	328.065	30,876.158
Broadwell-EP VM	1,543.815	50,005.289	1,229.626	102,610.375
STM32F407	–	2,075,202	89,911	5,294,305

Hosted wrapper latencies range from 10.0 to 50.0 μs for $\mathcal{C}_{384}$ and 19.9 to 102.6 μs for $\mathcal{C}_{320}$; the corresponding Cortex-M4 latencies are 2.08 and 5.29 ms.

To measure the effect on a complete cryptosystem, we compare three decoder integrations in ANSA-IBKEM1 and ANSA-IBKEM2, both of which use $\mathcal{C}_{384}$ with radius 14. The three paths use the conventional clean-word fast path, our complete table-free wrapper with randomization enabled, and a fixed-address WSR baseline at matching code parameters.

All variants share a source tree, GCC 12.2 with `-O3 -march=native -mavx2`, and an AVX2 Montgomery NTT, and pass scheme-level conformance tests. Measurements on the physical i7-13700 use six balanced blocks with rotating starting binaries and five one-second repetitions per invocation (30 observations per path). We report latency as the reciprocal of the pooled median throughput; 95% bootstrap intervals use 50,000 hierarchical draws. The timed wrapper includes randomness generation.

Table 9: Compressed ANSA-IBKEM decapsulation on a physical Intel i7-13700, in microseconds. Latencies are rounded; differences and percentage overheads use unrounded values. Overhead is relative to the functional but timing-leaky clean-word fast path; brackets give the 95% bootstrap interval for the complete-wrapper overhead.

Decoder path	ANSA-IBKEM1		ANSA-IBKEM2	
	latency (μs)	overhead	latency (μs)	overhead
Clean-word fast path (leaky)	26.2	–	87.1	–
Table-free complete wrapper	39.9	52.0% [51.8, 52.4]%	100.7	15.7% [15.4, 16.0]%
Fixed-address WSR baseline	1,015.4	3,769%	1,093.2	1,156%

Relative to the fast path, the complete wrapper adds 13.6 and 13.7 μs, whereas the fixed-address WSR baseline adds 989.2 and 1,006.2 μs. Decapsulation with our complete wrapper is 25.46× faster for ANSA-IBKEM1 (95% CI 25.39–25.68) and 10.85× faster for ANSA-IBKEM2 (95% CI 10.82–10.89) than with the fixed-address WSR baselines. The roughly one-millisecond WSR cost has a smaller relative effect in ANSA-IBKEM2 because the rest of its decapsulation path is longer. As a control for unrelated build variation, compressed encapsulation throughput differs across the three builds by 0.31% for ANSA-IBKEM1 and 0.30% for ANSA-IBKEM2.

5 Related work

Conventional BCH software uses table-based field arithmetic and RWD polynomial work [CS09]. WSR's ConSiBMA is the closest protected design: we inherit its paired recurrence and degree rule but replace protected field and shift accesses with fixed polynomial-basis circuits and public coefficient-pair relations [WSR20]. Tables 1 and 2 summarize

these changes and their computational costs.

McBits illustrates portable constant-time decoding with bitsliced Boolean field arithmetic [BCS13]. Our cores apply per-element polynomial-basis arithmetic within WSR's paired word-RAM decoding schedule.

Relational tools such as ct-verif, BINSEC/REL, and FaCT analyze or compile constant-time properties [ABB+16, DBR20, CSJ+19]. Our traces and audits provide evidence for specific binaries, not relational proofs. Compilers need not preserve constant-time intentions absent from C semantics [SCA18]. Random-codeword masking leaves BCH syndromes invariant [MSS13, RHHM17]; our optional cyclic shift is not a shuffle of decoder operations. Timing screens follow dudect-style mean tests [RBV17].

6 Conclusion

Algebraic index elimination realizes WSR's paired BCH recurrence through fixed computations over public addresses, replacing field-table scans, encoder-row selection, and indexed locator shifts. The deterministic construction preserves bounded-distance correctness and source-level trace regularity for primitive narrow-sense BCH parents and their systematic linear subcodes, including noncyclic ones.

For the investigated code parameters and compiler settings on the AArch64 VM, the table-free cores use less combined code and read-only storage than matched fixed-address WSR baselines. Code size alone is larger with `-O3` and smaller with `-Os`; estimated peak stack remains at most 464 bytes higher, while read-only storage is at least 5,152 bytes smaller. With `-O3`, the deterministic cores on the Intel i7-13700 are 35.7–82.4× faster than the fixed-address WSR baselines at WSR's published parameter sets. With our complete wrapper, ANSA-IBKEM1 and ANSA-IBKEM2 decapsulation is respectively 25.5× and 10.9× faster than with fixed-address WSR baselines at matching code parameters.

Source-level regularity alone does not settle the implementation's timing behavior. Apple Clang 21 turned ordinary C masks into RWD branches; a compiler-opaque primitive prevented this transformation in the audited builds. On M5, the tested scalar binaries pass the reported mean-timing screens with DIT enabled and show timing differences with DIT disabled. The microarchitectural cause remains unresolved, and the timing evidence is specific to the tested binaries and configurations.

These results support BCH as a viable error-correction choice for post-quantum KEMs: it can reduce bandwidth through ciphertext compression while admitting efficient constant-time decoding under the stated compiler and processor assumptions.

Use of ChatGPT. The authors used OpenAI's ChatGPT for language editing, code review, experiment automation, and review of the mathematical exposition. They verified all resulting text, proofs, implementations, and experimental claims and take full responsibility for the work.

References

[ABB+16] José Bacelar Almeida, Manuel Barbosa, Gilles Barthe, François Dupressoir, and Michael Emmi. Verifying constant-time implementations. In *25th USENIX Security Symposium*, pages 53–70. USENIX Association, 2016.

[AMAB+20] Carlos Aguilar Melchor, Nicolas Aragon, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Edoardo Persichetti, and Gilles Zémor. Hamming Quasi-Cyclic (HQC). NIST Post-Quantum Cryptography Standardization, Round 3 submission, 2020.

[Arm] Arm Limited. Arm architecture reference manual for a-profile architecture. Arm DDI 0487, https://developer.arm.com/documentation/ddi0487/latest. Section B1.5.6, "About PSTATE.DIT", accessed 7 August 2026.

[BCS13] Daniel J. Bernstein, Tung Chou, and Peter Schwabe. McBits: Fast constant-time code-based cryptography. In *Cryptographic Hardware and Embedded Systems – CHES 2013*, volume 8086 of *Lecture Notes in Computer Science*, pages 250–272. Springer, 2013.

[Chi64] Robert T. Chien. Cyclic decoding procedures for Bose–Chaudhuri–Hocquenghem codes. *IEEE Transactions on Information Theory*, 10(4):357–363, 1964.

[CS09] Junho Cho and Wonyong Sung. Efficient software-based encoding and decoding of BCH codes. *IEEE Transactions on Computers*, 58(7):878–889, 2009.

[CSJ+19] Sunjay Cauligi, Gary Soeller, Brian Johannesmeyer, Fraser Brown, Riad S. Wahby, John Renner, Benjamin Grégoire, Gilles Barthe, Ranjit Jhala, and Deian Stefan. FaCT: A DSL for timing-sensitive computation. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 174–189. ACM, 2019.

[CX26] Zhaohui Cheng and Kaixin Xiong. ANSA-IBKEM: Practical quantum-safe identity-based key encapsulation via annular NTRU trapdoors and standardized PQC arithmetic reuse. Cryptology ePrint Archive, Report 2026/1449, 2026.

[DBR20] Lesly-Ann Daniel, Sébastien Bardin, and Tamara Rezk. BINSEC/REL: Efficient relational symbolic execution for constant-time at binary-level. In *2020 IEEE Symposium on Security and Privacy*, pages 1021–1038. IEEE, 2020.

[DTVV19] Jan-Pieter D'Anvers, Marcel Tiepelt, Frederik Vercauteren, and Ingrid Verbauwhede. Timing attacks on error correcting codes in post-quantum schemes. In *Proceedings of the ACM Workshop on Theory of Implementation Security*, pages 2–9. ACM, 2019.

[DVV18] Jan-Pieter D'Anvers, Frederik Vercauteren, and Ingrid Verbauwhede. On the impact of decryption failures on the security of LWE/LWR based schemes. Cryptology ePrint Archive, Report 2018/1089, 2018.

[GJJR11] Gilbert Goodwill, Benjamin Jun, Josh Jaffe, and Pankaj Rohatgi. A testing methodology for side-channel resistance validation. In *NIST Non-Invasive Attack Testing Workshop*, 2011.

[GMGL24] Guillaume Goy, Julien Maillard, Philippe Gaborit, and Antoine Loiseau. Single trace HQC shared key recovery with SASCA. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2024(2):64–87, 2024.

[GNJ26] Qian Guo, Denis Nabokov, and Thomas Johansson. Unlocking the true potential of decryption failure oracles: A hybrid adaptive-LDPC attack on ML-KEM using imperfect oracles. Cryptology ePrint Archive, Report 2026/070, 2026.

[Ham19] Mike Hamburg. Post-quantum cryptography proposal: ThreeBears. NIST Post-Quantum Cryptography Standardization, Round 2 submission, 2019.

[HP03] W. Cary Huffman and Vera Pless. *Fundamentals of Error-Correcting Codes*. Cambridge University Press, 2003.

[LLZ+18] Xianhui Lu, Yamin Liu, Zhenfei Zhang, Dingding Jia, Haiyang Xue, Jingnan He, Bao Li, and Kunpeng Wang. LAC: Practical ring-LWE based public-key encryption with byte-level modulus. Cryptology ePrint Archive, Report 2018/1009, 2018.

[LWC+26] Xianhui Lu, Jiabo Wang, Dongshu Cai, Yijian Liu, Yamin Liu, Lei Bi, Dingding Jia, Jingnan He, Ying Liu, Ziyao Liu, Yu Zhang, and Bo Pang. PolarLAC: Polar code-enhanced lattice-based cryptosystem. NICCS Public-Key Cryptographic Algorithms, Round 1 candidate, 2026.

[Mas69] James L. Massey. Shift-register synthesis and BCH decoding. *IEEE Transactions on Information Theory*, 15(1):122–127, 1969.

[MSS13] Dominik Merli, Frederic Stumpf, and Georg Sigl. Protecting PUF error correction by codeword masking. Cryptology ePrint Archive, Report 2013/334, 2013.

[PRJ+23] Thales Paiva, Prasanna Ravi, Dirmanto Jap, Shivam Bhasin, Sayan Das, and Anupam Chattopadhyay. Et tu, brute? SCA assisted CCA using valid ciphertexts: A case study on HQC KEM. Cryptology ePrint Archive, Report 2023/1626, 2023.

[RBV17] Oscar Reparaz, Josep Balasch, and Ingrid Verbauwhede. Dude, is my code constant time? In *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1697–1702. IEEE, 2017.

[RHHM17] Mélissa Rossi, Mike Hamburg, Michael Hutter, and Mark E. Marson. A side-channel assisted cryptanalytic attack against QcBits. In *Cryptographic Hardware and Embedded Systems – CHES 2017*, volume 10529 of *Lecture Notes in Computer Science*, pages 3–23. Springer, 2017.

[SCA18] Laurent Simon, David Chisnall, and Ross Anderson. What you get is what you C: Controlling side effects in mainstream C compilers. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 1–15. IEEE, 2018.

[SRSWZ21] Thomas Schamberger, Julian Renner, Georg Sigl, and Antonia Wachter-Zeh. A power side-channel attack on the CCA2-secure HQC KEM. In *Smart Card Research and Advanced Applications – CARDIS 2020*, volume 12609 of *Lecture Notes in Computer Science*, pages 119–134. Springer, 2021.

[WSR19] Matthew Walters and Sujoy Sinha Roy. Constant-time BCH implementation. GitHub repository, 2019. Accessed 10 August 2026.

[WSR20] Matthew Walters and Sujoy Sinha Roy. Constant-time BCH error-correcting code. In *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2020. Full version: IACR Cryptology ePrint Archive, Report 2019/155, 2019, https://eprint.iacr.org/2019/155. Code: https://github.com/mjw553/Constant_BCH.