Codeword Masking Can Be Harmful Under Replay Attacks on HQC

Jaeho Jeon[1], Yongseong Park[1], Jaeyeon Lee[1], Suseong Lee[1], Donghyeon Kim[2], and Young-Sik Kim[1(✉)]

[1] Department of Electrical Engineering and Computer Science, DGIST, Daegu, Republic of Korea
{dgwogh,reo91004,ljy8733,mercury,ysk}@dgist.ac.kr
[2] School of Undergraduate Studies, DGIST, Daegu, Republic of Korea
dhkim200426@dgist.ac.kr

Abstract. HQC, selected by NIST for standardization in 2025, was recently shown vulnerable to a replay attack recovering $\mathbf{v}-\mathbf{u}\cdot\mathbf{y}$, from which the long-term secret $\mathbf{y}$ was shown to be recoverable, and against which *codeword masking* was proposed as the countermeasure. We revisit both and demonstrate that the countermeasure yields an opposite effect: on the target build, the unmasked replay attack is ineffective, whereas introducing codeword masking paradoxically enables its success. We present a new leakage venue in HQC, in the Reed–Muller encoder, and apply to it the post-decoding idea of recent work, turning HQC's own error correction into a side-channel budget that recovers the message: a single trace suffices, with only ∼80 profiling traces, and we confirm the same across the official implementations (the NIST reference and optimized code, and PQClean) at both `-O3` and `-Os`. We then improve the replay attack on $\mathbf{v}-\mathbf{u}\cdot\mathbf{y}$, raising its per-bit recovery probability; at the optimization level we study, that is still not enough for a practical attack. Codeword masking, adopted as the defense, supplies what replay could not: the random share erases the ambiguity that identical replays leave behind, and the extrapolated probability of recovering the whole $\mathbf{v}-\mathbf{u}\cdot\mathbf{y}$ rises from 10^{-555} to 1.0. Increasing the masking order fails to mitigate the attack: we demonstrate identical recovery against a three-share implementation, without incurring additional profiling overhead for the adversary. Masking must therefore be designed against the optimization level and the device it is deployed on.

Keywords: HQC · Post-quantum cryptography · KEM · Side-channel analysis · Codeword masking · Reed–Muller encoding · Reed–Solomon decoding · Fujisaki–Okamoto transform

1 Introduction

1.1 Post-Quantum Cryptography and Message-Recovery Attacks

Shor's algorithm [50] breaks classical public-key schemes such as RSA [49] and ECC [41] in quantum polynomial time, prompting NIST's Post-Quantum Cryp-

tography (PQC) standardization process [9,14]. The 2022 selections deliberately span diverse hardness assumptions [1]: the lattice-based KEM CRYSTALS-Kyber (ML-KEM) [11], the signature schemes CRYSTALS-Dilithium (ML-DSA) [21] and Falcon (FN-DSA) [22], and the hash-based SPHINCS+ [8]. To broaden the assumption base beyond lattices, the fourth round retained the code-based candidates BIKE [4], Classic McEliece [7], and Hamming Quasi-Cyclic (HQC) [38], with HQC selected for standardization in 2025 [2].

NIST has stressed that practical security on embedded, resource-constrained platforms matters as much as theoretical robustness [1], and the standardized algorithms have since been extensively evaluated against side-channel and implementation attacks: secret-key recovery on Kyber [45,44,30], on Dilithium [48,53,54] and Falcon [37,33,29], alongside masking and shuffling countermeasures [12,40,13].

While this early literature mostly targets long-term secret keys, recovering the ephemeral message of a single decapsulation is equally damaging: in KEMs built on the Fujisaki–Okamoto (FO) transform [23] the message reconstructs the shared key directly from the public ciphertext, and even *partial* message recovery escalates to long-term-key recovery via plaintext-checking attacks [52,47]. Message-recovery power and EM attacks have been demonstrated on both lattice-based KEMs (Kyber, Saber, FrodoKEM [51,42,10]) and code-based KEMs (Classic McEliece, HQC [34,16,28]), with the leakage typically arising in the structurally regular arithmetic that FO-style decapsulation re-executes deterministically.

1.2 Survey of Prior Side-channel Analyses on HQC

We organize prior side-channel work on HQC (summarized in Figure 1), studied intensively since its 2020 switch from a tensor-product (repetition $\times$ BCH) inner code [38] to the concatenated Reed–Muller/Reed–Solomon construction [24], along two axes: direct recovery of the encapsulated message (and hence the shared key) from one or a few traces, and oracle-based recovery of the long-term secret key through crafted decapsulation queries.

A first thread targets specific operations of the Reed–Muller and Reed–Solomon decoders during decryption. Goy et al. [27] mount a chosen-ciphertext template attack on the RM decoder's Hadamard transform, recovering key bits one at a time; the same authors later turn to the RS decoder's `gf_mul` with a SASCA-based single-trace attack [28], extended to the RS encoder re-executed during Fujisaki–Okamoto re-encryption. Maillet et al. [36] attack the first stage of the optimized RM decoder, `expand_and_sum`, which folds the duplicated copies of each Reed–Muller codeword into a single accumulator before the transform runs; they read it under a Hamming-distance model. Paiva et al. [43] extend the same routine, together with the subsequent `find_peaks`, to a setting using only *valid* ciphertexts. Dong et al. [20] give two complementary single-trace attacks: a passive VC-KRA-MDPC that feeds noisy `expand_and_sum` leakage to an MDPC-style belief-propagation decoder, and an active OT-FDA-CK that builds a full-decryption oracle from the RM decoder outputs (read through the syndrome `gf_mul`) and probes the linear combination $x + y$; both recover the

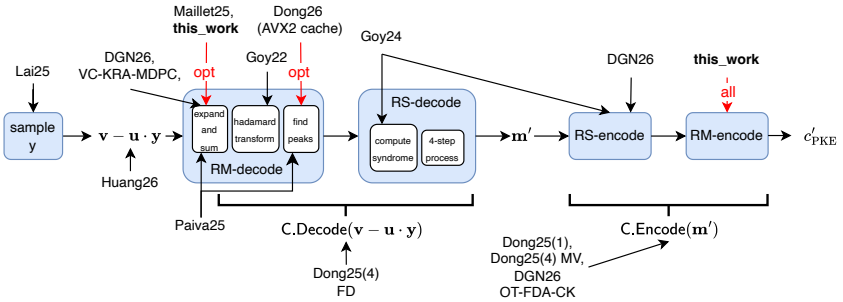


Fig. 1. Side-channel attack venues across the HQC decapsulation pipeline. Labels: *Goy22* [27], *Goy24* [28], *Maillet25* [36], *Paiva25* [43], *Huang26* [32], *Dong25(1)* [18], *Dong25(4) (FD/MV)* [17], *DGN26 (VC-KRA-MDPC, OT-FDA-CK)* [20], *Dong26 (AVX2 cache)* [19], *Lai25* [35]. Red arrows mark venues attacked in the optimized implementation. *This work* targets the inner Reed–Muller encoder.

long-term key. Dong and Guo [19] instead reach `find_peaks` by cache timing on the AVX2 build. Finally, Lai et al. [35] give a ciphertext-independent timing attack on the key re-sampling and the $u \cdot y$ multiplication preceding the pipeline, and Huang et al. [32] a DPA on the `vect_add` that realizes $\mathbf{v} - \mathbf{u} \cdot \mathbf{y}$ at the RM decoder input.

A second thread refines plaintext-checking (PC) oracle attacks through offline templates. Dong and Guo's OT-PCA [18] builds an offline template from the public decoder, cutting the per-query cost by orders of magnitude; they then extend it to multi-value (MV-PC) and full-decryption (FD) oracles [17], instantiated from leakage spanning both C.Encode during re-encryption and the RS syndrome `gf_mul` loop.

To our knowledge, no published attack targets the inner Reed–Muller *encoder* that the FO transform re-executes during decapsulation. We identify this venue as a structurally regular, cross-implementation leakage source and analyze it alongside the `expand_and_sum` venue within a single post-decoding framework; the contributions are summarized in Section 1.3.

1.3 This Work and Contributions

We make three contributions to the side-channel analysis of HQC.

1. **A New RM-encoder Venue and A Quantified Post-Decoding Lift.** We are the first to identify the Reed–Muller *encoder* as a side-channel venue in HQC. It runs during encapsulation and again inside the Fujisaki–Okamoto re-encryption that decapsulation performs. We examined three implementations: the NIST reference (identical to the HQC team's GitLab code), the NIST optimized version, and PQClean. In each, it expands the message one bit at a time, leaking each bit strongly and at its own time offset. It is also

exactly where the post-decoding lift of Goy et al. [28] applies: HQC carries its own error correction, provisioned for a low decryption-failure rate, and that budget absorbs side-channel noise as readily as channel noise. Feeding the noisy per-byte estimates to HQC's *own* public Reed–Solomon decoder, we recover the 16-byte message bit-exactly from a *single* trace at 100%, with a per-byte CNN profiled on as few as ∼80 traces (STM32F415). The result is not particular to one build: all three codebases reach 100% recovery at both `-Os` and `-O3`, there from 400 profiling traces per build.

2. **Per-trace Soft Voting with Multi-POI Selection.** We extend the replay attack of Maillet et al. [36] with a broader leakage hypothesis and aggregation rule. Their single-POI shift-register model holds under `-O2`, but under `-O3` the leakage moves to the accumulator read–modify–write, a compilation effect rather than a device one. To cover both, we select points of interest comprehensively (multi-POI) and aggregate the replays by per-trace *soft voting*, which outperforms the signal-averaging approach of Maillet et al. in mitigating measurement noise and accommodating input variability. Per-bit accuracy on $\mathbf{v} - \mathbf{u} \cdot \mathbf{y}$ rises from 0.747 to 0.933 (+18.6 pp).
3. **Codeword Masking Is Harmful under Replay Attacks, Not Defensive.** Combining (1) and (2) leads to the paper's central result: the countermeasure is harmful rather than protective. In (2) we rebuilt the POI strategy around the `-O3` assembly and raised per-bit accuracy, but that is not enough. At $R = 500$, recovering a 384-bit `expand_and_sum` block bit-exactly still succeeds with probability only $\approx 10^{-12.1}$, and a full $\mathbf{v} - \mathbf{u} \cdot \mathbf{y}$ is 46 such blocks, so $\approx 10^{-555}$: on this build the attack does not work. Applying the codeword masking of Maillet et al. [36] is what makes it work. The masked decoder processes $(\mathbf{v} - \mathbf{u} \cdot \mathbf{y}) + \mathbf{m}'\mathbf{G}$, which re-invokes the encoder of (1), so the share $\mathbf{m}'$ is recovered directly from its own trace; and because $\mathbf{m}'$ is refreshed at every decapsulation, the shares vary across replays and average out the accumulator ambiguity that identical replays cannot touch. With masking in place, $R \approx 500$ replays recover the 384-bit block with probability 1.0, and the full $\mathbf{v} - \mathbf{u} \cdot \mathbf{y}$ with it. The same holds at third order: recovering three shares costs no more profiling than one, and the key recovery is unchanged. Codeword masking must therefore be reconsidered before deployment; the mechanism is not HQC-specific, applying wherever a single-trace-recoverable target is re-encoded under per-decapsulation random shares.

A full implementation of the analysis pipeline used here is available at `https://github.com/my-rooftop/codeword_masking_can_be_harmful`.

2 Background

2.1 HQC Scheme

We summarize the HQC-PKE algorithms at a high level, omitting implementation details related to XOF initialization, seed expansion, and the implementation-specific compression/truncation of ciphertext components. We focus instead on the algebraic structure relevant to oracle construction and message leakage.

HQC-PKE key generation samples two sparse secret vectors x, y and a dense public vector h, and computes $s = x + h \cdot y$. The public key is $\mathsf{pk} = (h, s)$ and the secret key contains $\mathsf{sk} = (x, y)$. We refer to the HQC specification [24] for the full pseudocode.

To encrypt a message $\mathbf{m}$ under $\mathsf{pk} = (h, s)$ (Algorithm 1), the algorithm samples sparse ephemeral values (r_1, r_2, e) and outputs a ciphertext $c_{\mathrm{PKE}} = (u, v)$ where

$$u = r_1 + h \cdot r_2, \qquad v = \mathsf{C.Encode}(\mathbf{m}) + s \cdot r_2 + e.$$

Here, $\mathsf{C.Encode}(\cdot)$ denotes HQC's concatenated RS/RM encoding, whose implementation is analyzed in Section 3.

Algorithm 1 HQC-PKE.Encrypt (simplified)

Require: Public key $\mathsf{pk} = (h, s)$, message $\mathbf{m}$
Ensure: Ciphertext $c_{\mathrm{PKE}} = (u, v)$
1: Sample $r_1, r_2, e \leftarrow \mathsf{SampleFixedWeightVect}(\mathcal{R}_\omega)$
2: $u \leftarrow r_1 + h \cdot r_2$
3: $v \leftarrow \mathsf{C.Encode}(\mathbf{m}) + s \cdot r_2 + e$
4: **return** $c_{\mathrm{PKE}} = (u, v)$

Given $c_{\mathrm{PKE}} = (u, v)$ and secret y, decryption computes

$$\mathbf{m} = \mathsf{C.Decode}\big(v - u \cdot y\big),$$

where $\mathsf{C.Decode}(\cdot)$ denotes HQC's concatenated RS/RM decoding. This affine structure will also be relevant for understanding how decoding acts on noisy symbol estimates and why message-dependent behavior in the encoding and decoding pipeline can have security consequences. The intermediate value $v - u \cdot y$ is itself the side-channel target studied in Sections 4 and 5.

HQC-KEM applies an FO-style transform on top of HQC-PKE. Decapsulation first computes a candidate message $\mathbf{m}' \leftarrow \mathsf{HQC-PKE.Decrypt}(\mathsf{sk}_{\mathrm{PKE}}, c_{\mathrm{PKE}})$, then deterministically derives both the candidate shared key and the re-encryption randomness as $(K', \theta') \leftarrow G(H(\mathsf{pk}_{\mathrm{KEM}}) \,\|\, \mathbf{m}' \,\|\, \mathsf{salt})$, and re-encrypts $\mathbf{m}'$ as $c'_{\mathrm{PKE}} \leftarrow \mathsf{HQC-PKE.Encrypt}(\mathsf{pk}_{\mathrm{KEM}}, \mathbf{m}', \theta')$ to validate the ciphertext. If $c'_{\mathrm{PKE}} = c_{\mathrm{PKE}}$ (and $\mathbf{m}' \neq \perp$), the output is $K = K'$; otherwise the implementation returns a rejection key $\bar{K} \leftarrow J(H(\mathsf{pk}_{\mathrm{KEM}}) \,\|\, \sigma \,\|\, c_{\mathrm{KEM}})$. Crucially, the deterministic re-encryption re-executes $\mathsf{C.Encode}(\mathbf{m}')$, which is the leakage point studied in this work.

2.2 RS / RM Encoding in HQC (systematic form)

As introduced in Section 2.1, HQC encodes the 128-bit encapsulation message $\mathbf{m}$ using a concatenated error-correcting code $\mathsf{C.Encode}(\cdot)$: an outer Reed–Solomon (RS) code over $\mathbb{F}_{256}$ followed by an inner (binary) Reed–Muller (RM) code [38].

Algorithm 2 HQC-KEM.Decaps (FO transform, simplified)

Require: Secret key $\mathsf{sk}_{\mathrm{KEM}}$, ciphertext c_{KEM}
Ensure: Shared key K
1: $\mathbf{m}' \leftarrow \mathsf{HQC-PKE.Decrypt}(\mathsf{sk}_{\mathrm{PKE}}, c_{\mathrm{PKE}})$
2: $(K', \theta') \leftarrow G\big(H(\mathsf{pk}_{\mathrm{KEM}}) \,\|\, \mathbf{m}' \,\|\, \mathsf{salt}\big)$
3: $c'_{\mathrm{PKE}} \leftarrow \mathsf{HQC-PKE.Encrypt}(\mathsf{pk}_{\mathrm{KEM}}, \mathbf{m}', \theta')$ ▷ re-encryption
4: $\bar{K} \leftarrow J\big(H(\mathsf{pk}_{\mathrm{KEM}}) \,\|\, \sigma \,\|\, c_{\mathrm{KEM}}\big)$ ▷ rejection key
5: **if** $\mathbf{m}' \neq \perp$ **and** $c'_{\mathrm{PKE}} = c_{\mathrm{PKE}}$ **then**
6: $\quad K \leftarrow K'$
7: **else**
8: $\quad K \leftarrow \bar{K}$
9: **end if**
10: **return** K

Intuitively, the RS layer protects the message at the *symbol* (byte) level, while the RM layer protects each RS symbol at the *bit* level before transmission.

For HQC-128, the RS encoder takes the 128-bit message $\mathbf{m}$ (i.e., $k = 16$ bytes) and produces an RS codeword of length $n_1 = 46$ bytes, expanding the payload from $16 \times 8 = 128$ bits to $46 \times 8 = 368$ bits. HQC uses a *systematic* RS encoder, meaning the message bytes appear *verbatim* in fixed positions of the RS codeword:

$$\mathtt{cdw}[30..45] = \mathbf{m}[0..15], \qquad \mathtt{cdw}[0..29] \text{ are parity symbols.}$$

The systematic placement of $\mathbf{m}$ in $\mathtt{cdw}[30..45]$ is also relevant from an attack perspective: a direct message-recovery attack would need to recover all 16 systematic bytes, whereas an attack that recovers the full 46-byte codeword and relies on the outer RS decoder to absorb residual symbol errors operates under a substantially softer success criterion. We make this contrast quantitative in Section 3.4.

After RS encoding, HQC applies the inner RM encoder independently to each RS symbol. Concretely, HQC uses the first-order RM code RM$(1, 7)$, which maps one 8-bit input symbol to a 128-bit binary codeword. HQC then applies a duplication factor $d = 3$ (bit repetition), so each RS symbol becomes a $128 \cdot 3 = 384$-bit block. Thus, encoding the full RS codeword of 46 symbols yields a total of

$$46 \times 384 = 17{,}664 \text{ bits},$$

which naturally decomposes into a parity block ($30 \times 384 = 11{,}520$ bits) and a message block ($16 \times 384 = 6{,}144$ bits), mirroring the systematic RS partition.

In HQC-PKE encryption, the binary codeword $\mathsf{C.Encode}(\mathbf{m})$ is then combined with ciphertext-dependent terms and sparse noise (error injection). As a result, the receiver observes a noisy version of the concatenated codeword, which must be corrected by $\mathsf{C.Decode}(\cdot)$ at decryption time.

2.3 RM Decoding (`expand_and_sum`) and RS Post-Decoding

The decoder $\mathsf{C.Decode}(\cdot)$ inverts the inner Reed–Muller stage first and then the outer Reed–Solomon stage.

The base inner code is the first-order Reed–Muller code $\mathrm{RM}(1,7)$ with parameters $[128, 8, 64]$ over $\mathbb{F}_2$. HQC duplicates each base codeword d times to obtain a $[128d,\ 8,\ 64d]$ block, with $d = 3$ for HQC-128 ($[384, 8, 192]$) and $d = 5$ for HQC-192 and HQC-256 ($[640, 8, 320]$). Decoding proceeds in two phases. First, the implementation routine `expand_and_sum` collapses the duplicated block into a single 128-element integer accumulator by adding the ± 1-mapped copies position by position. Second, a fast Hadamard transform performs maximum-likelihood decoding of the resulting length-128 sequence against the 256 RM codewords, after which a `find_peaks` routine returns the index of the maximum-magnitude transform coefficient as the decoded byte. The duplication amplifies the per-block minimum distance from 64 to $64d$, so the inner decoder absorbs up to $\lfloor (64d - 1)/2 \rfloor$ bit errors per block: 95 errors for HQC-128 and 159 errors for HQC-192 and HQC-256. The multi-copy summing structure of `expand_and_sum` is the origin of the per-bit access pattern that we exploit in Section 4.

The inner decoder outputs one byte per RM block, and the resulting byte sequence is handed to a shortened Reed–Solomon decoder over $\mathbb{F}_{256}$. The RS decoder first forms the error syndromes of the received word (the `compute_syndromes` routine), then runs Berlekamp's algorithm followed by an additive-FFT root-finding step. All $\mathbb{F}_{256}$ multiplications in the RS encoder and decoder, including the systematic encoder of Section 2.2 and the syndrome computation, are performed by a single Galois-field multiplication primitive `gf_mul`, which several of the prior attacks surveyed in this paper target directly. Each shortened code RS-S$i[\, n_1, k_1, \delta\,]$ corrects up to δ symbol (byte) errors, with per-instance parameters RS-S1 $= [46, 16, 15]$ for HQC-128, RS-S2 $= [56, 24, 16]$ for HQC-192, and RS-S3 $= [90, 32, 29]$ for HQC-256. This outer stage is load-bearing for HQC's decoding chain: the decoding-failure-rate (DFR) targets of $< 2^{-128}$, $< 2^{-192}$, and $< 2^{-256}$ for HQC-128, HQC-192, and HQC-256 are realized by relying on the RS decoder to absorb the residual symbol errors that survive the inner RM stage. From the attacker's side, the same byte-level correction radius δ provides an intrinsic error tolerance budget under which noisy per-byte estimates can be lifted to the exact RS codeword, the structural resource exploited by the post-decoding lift of Section 3.4.

3 Message Recovery via RM-Encoder Leakage and RS Post-Decoding

A common structural ingredient runs through several recent side-channel analyses of HQC and underpins the attack of this section: noisy per-byte estimates obtained by side-channel inference are turned into an exact RS codeword by *re-using* HQC's public Reed–Solomon decoder as a post-correction primitive. We call this the *post-decoding lift principle*. Goy et al. [28] put this principle

to work for HQC shared-key recovery: instead of recovering the RS codeword in full from the side channel, they recover it only in part (the first codeword byte never passes through the multiplication their templates target, and is left unrecovered) and let the post-decoding lift restore the shared key reliably. The resulting pipeline is tolerant to measurement noise, and it is the first to carry the idea of Goy, Loiseau, and Gaborit [26] from a theoretical vulnerability to a practical attack; Baïsse et al. [5] reuse it again.

We exploit a new leakage venue, a weakness in the byte-wise processing of the Reed–Muller encoder executed during FO re-encryption, and apply the same post-decoding lift principle to recover the message reliably from a small profiling-trace budget.

This section presents our first attack pipeline: single-trace message recovery from the Reed–Muller (RM) encoder executed during Fujisaki–Okamoto (FO) re-encryption. We first identify a *structured* instruction-level leakage source in the optimized RM encoder (Sections 3.1–3.2) and characterize its per-bit exploitability (Section 3.3). We then formalize the central robustness property of the attack pipeline as the *RS post-decoding lift theorem* (Section 3.4), and empirically validate it on STM32F415 (Section 3.5).

All code-level observations and the TVLA assessment below are taken from the official HQC reference implementation [24,31] at `-O3`, the build NIST received (*NIST-ref*). Section 3.6 confirms the same leakage on the NIST optimized implementation (*NIST-opt*) and on PQClean, at both `-Os` and `-O3`.

3.1 The RM-Encoder Venue: BIT0MASK and `sbfx` Expansion

RS encoding executes first and preserves message bytes verbatim. Recall from Section 2.2 that HQC uses a *systematic* RS encoder: the 16-byte message is copied verbatim into positions 30..45 of the 46-byte RS codeword, while positions 0..29 hold parity symbols. The HQC reference implementation realizes this with a single `memcpy` (line 11 of Listing 1.1) that places the message bytes at the systematic offset; the `memcpy` that follows only packs the finished codeword into the output buffer.

```
1  void reed_solomon_encode(uint64_t *cdw, const uint64_t *msg) {
2      /* initialize generator-poly coefficients and scratch buffers */
3
4      uint8_t  msg_bytes[PARAM_K] = {0};
5      uint8_t  cdw_bytes[PARAM_N1] = {0};
6
7      memcpy(msg_bytes, msg, PARAM_K);
8
9      /* RS parity computation: fills cdw_bytes[0 .. PARAM_N1-PARAM_K-1] (
           omitted) */
10
11     memcpy(cdw_bytes + PARAM_N1 - PARAM_K, msg_bytes, PARAM_K); /* systematic
           copy */
12     memcpy(cdw, cdw_bytes, PARAM_N1);
13 }
```

Listing 1.1. Systematic Reed–Solomon encoding in HQC (excerpt). The `memcpy` on line 11 places the message bytes verbatim in positions `[PARAM_N1−PARAM_K, PARAM_N1)` of the codeword; the one on line 12 merely packs the byte array into the output buffer.

As a result, when the encoder pipeline reaches the inner RM stage, the target secret bytes appear unaltered as a contiguous suffix of `cdw_bytes`, and the remaining 30 bytes carry parity, which, as we exploit in Section 3.4, is precisely what enables the RS post-decoding lift.

After RS encoding, the 46-byte codeword `cdw_bytes` is passed to the inner RM encoder. HQC encodes each byte independently using the first-order Reed–Muller code RM(1, 7), which maps an 8-bit input to a 128-bit codeword $\mathbf{c} = \mathbf{m}\,G$ over $\mathbb{F}_2$. The generator matrix G has the compact 32-bit periodic structure

```
0:  aaaaaaaa aaaaaaaa aaaaaaaa aaaaaaaa
1:  cccccccc cccccccc cccccccc cccccccc
2:  f0f0f0f0 f0f0f0f0 f0f0f0f0 f0f0f0f0
3:  ff00ff00 ff00ff00 ff00ff00 ff00ff00
4:  ffff0000 ffff0000 ffff0000 ffff0000
5:  ffffffff 00000000 ffffffff 00000000
6:  ffffffff ffffffff 00000000 00000000
7:  ffffffff ffffffff ffffffff ffffffff
```

Rather than computing $\mathbf{m}\,G$ via explicit bit-matrix multiplication, the HQC reference implementation uses a word-level expansion driven by the macro $\texttt{BITOMASK}(x) \triangleq -((x)\,\&\,1)$, which expands a single message bit into a full 32-bit mask of either `0x00000000` or `0xFFFFFFFF`. Listing 1.2 reproduces the relevant routine:

```
#define BITOMASK(x) (int32_t)(-((x) & 1))

typedef union {
    uint8_t  u8[16];    /**< Byte-wise access  */
    uint32_t u32[4];    /**< Word-wise access  */
} rm_codeword_t;

void encode(rm_codeword_t *word, int32_t message) {
    int32_t first_word;
    first_word  = BITOMASK(message >> 7);
    first_word ^= BITOMASK(message >> 0) & 0xaaaaaaaa;
    first_word ^= BITOMASK(message >> 1) & 0xcccccccc;
    first_word ^= BITOMASK(message >> 2) & 0xf0f0f0f0;
    first_word ^= BITOMASK(message >> 3) & 0xff00ff00;
    first_word ^= BITOMASK(message >> 4) & 0xffff0000;
    word->u32[0] = first_word;
    first_word ^= BITOMASK(message >> 5);
    word->u32[1] = first_word;
    first_word ^= BITOMASK(message >> 6);
    word->u32[3] = first_word;
    first_word ^= BITOMASK(message >> 5);
    word->u32[2] = first_word;
}
```

Listing 1.2. Reed–Muller encoding routine in HQC (excerpt).

When compiled with `arm-none-eabi-gcc` at `-O3`, the compiler realizes each `BITOMASK` expansion with a single `sbfx` (Signed Bit-Field Extract) instruction, `sbfx rX, r3, #i, #1`. It extracts bit i of the message byte and sign-extends it into a 32-bit register value of either `0x00000000` or `0xFFFFFFFF`. The subsequent `AND` operations with large structured constants (`0xaaaaaaaa`, `0xcccccccc`, ...) and the `EOR` chain that accumulates them into `first_word` produce intermediate register values whose Hamming weight is strongly determined by the individual

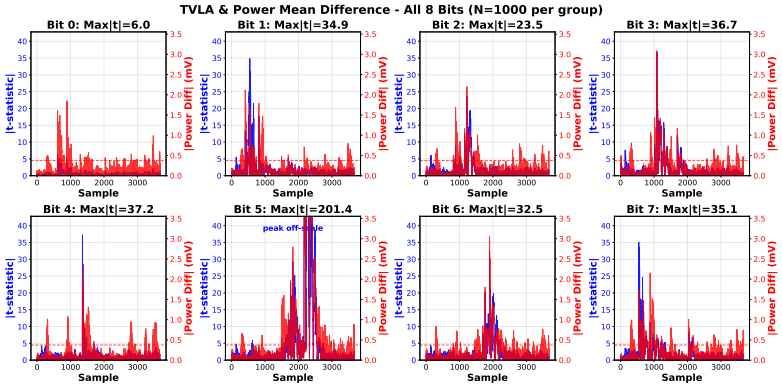


Fig. 2. TVLA of the HQC RM encoder on STM32F415 (1,000 traces per bit value, other bits random). Each panel corresponds to one input bit $b \in \{0, \ldots, 7\}$. Blue/left axis: Welch t statistic; red/right axis: mean power difference.

message bits. Because this same instruction sequence is executed 46 times per $\mathsf{C.Encode}(\mathbf{m}')$ invocation, once per RS symbol, the RM encoder provides a temporally separable, repeatable, and cross-implementation leakage source.

3.2 Empirical Leakage Assessment (TVLA)

To quantify the leakage of the RM-encoding routine, we collected power traces using a PicoScope 3403D in combination with a ChipWhisperer CW308 UFO board equipped with an STM32F415 target. The ChipWhisperer-Lite was used in parallel for synchronized triggering, ensuring stable trace alignment across measurements. To check whether the RM encoder produces statistically detectable leakage points, we ran a first-order Welch's t-test with a fixed-vs-fixed (FvF) partitioning, in the sense of the TVLA (Test Vector Leakage Assessment) methodology of Goodwill et al. [25]. For each target bit position $b \in \{0, \ldots, 7\}$ we formed two groups (1,000 traces with bit b fixed to 0 and 1,000 with bit b fixed to 1, the remaining bits random) and computed the per-sample mean power difference and the Welch t statistic. Time samples with $|t| > 4.5$ are taken as evidence of statistically significant first-order leakage.

Figure 2 shows the resulting t-test traces. We observe strong, temporally localized leakage points for every message bit. The peak t values are far above the conventional detection threshold ($|t| > 4.5$), confirming that the RM-encoder leakage is both strong and directly exploitable. Equally important, the leakage points for different bits occupy *distinct* time offsets, matching the sequential bit-processing order in Listing 1.2. This temporal separability is favorable for the attacker, as it enables independent per-bit discrimination with minimal interference from other bits.

Listing 1.2 applies the bits in the order $7 \to 0 \to 1 \to 2 \to 3 \to 4 \to 5 \to 6 \to 5$, not in numerical order, with three specific bits exhibiting substantially higher

leakage than the others. Bit 7 runs first and is the only bit applied without an `AND` mask, setting all 32 bits of `first_word` at once. Bit 5 is applied twice, once for `word[1]` and again to cancel itself for `word[2]`, so it leaks at two separate points. Bits 5 and 6 are XORed into a word that already carries bits 0–4, which leaves a larger Hamming-weight swing than the masked bits produce on their own.

3.3 Per-Byte Classification: 8-Bit Classifiers and the Brittleness of Direct Message Recovery

Building on the bit-level leakage characterized above, we trained eight independent binary classifiers, one per bit position, each shared across all 46 RM-encoded bytes. Each classifier outputs the posterior over the value of bit b given the trace segment surrounding its POI. All classifiers in this paper share one small 1-D CNN and one training recipe, set out in Appendix A; only the input window and the profiling budget differ across venues. The resulting per-bit accuracies on STM32F415 are

$$(89.8,\ 95.0,\ 93.5,\ 97.8,\ 91.8,\ 100.0,\ 99.5,\ 95.8)\,\% \quad \text{for } b = 0, 1, \ldots, 7,$$

with an arithmetic mean of about 95.4%. Although these individual accuracies appear promising, they do not directly ensure successful full-message recovery.

In practice, the primary metric of interest for an adversary is the recovery of an entire *byte* (all eight bits correct simultaneously) and ultimately of the entire 16-byte message. Under independence, the per-byte success probability is the product of the per-bit accuracies, which collapses to roughly 68%. Compounded over the 16 systematic message bytes, the full-message success rate is

$$P_{\mathrm{raw}}(p) \;=\; p^{16}, \tag{1}$$

which at $p \approx 0.68$ amounts to only about 0.22%. In other words, despite per-bit accuracies in the 90% range, raw 16-byte message recovery is essentially infeasible: a single residual bit error anywhere in the 128 systematic bits is enough to destroy the entire message.

3.4 The RS Post-Decoding Lift

The 0.22% raw recovery rate of Section 3.3 is an artifact of the attacker's recovery target, not of the leakage itself. We now formalize the observation that, by targeting the full 46-byte RS codeword instead of only the 16 systematic message bytes, the attacker can reuse the public RS decoder as a post-correction primitive, and inherits the same noise tolerance as the legitimate decoder.

Setup. The HQC RS code RS-S1 has parameters $(n_1, k, d_{\min}) = (46, 16, 31)$ over $\mathbb{F}_{256}$, so its bounded-distance correction capacity is $\delta = \lfloor (d_{\min} - 1)/2 \rfloor = 15$. Let $\hat{\mathbf{c}} \in \mathbb{F}_{256}^{46}$ denote the noisy 46-symbol RS codeword recovered by the attacker

from a single side-channel trace, and let $\mathbf{c}$ denote the ground-truth codeword. Write $E(\hat{\mathbf{c}}, \mathbf{c}) = |\{\, i : \hat{c}_i \neq c_i \,\}|$ for the number of erroneous RS symbols.

The public HQC RS decoder $\mathcal{D}_{\mathrm{RS}}$ reconstructs the original message $\mathbf{m}$ exactly whenever $E(\hat{\mathbf{c}}, \mathbf{c}) \leq \delta$.

Theorem 1 (RS Post-Decoding Lift). *Assume the attacker's per-symbol estimates are independent and identically distributed, with per-symbol correctness probability $p \in (0, 1]$. Then the probability that $\mathcal{D}_{\mathrm{RS}}(\hat{\mathbf{c}}) = \mathbf{m}$ is*

$$P_{\mathrm{RS}}(p) = \sum_{k=n_1-\delta}^{n_1} \binom{n_1}{k} p^{\,k} \, (1-p)^{\,n_1-k}, \qquad \textit{with } n_1 = 46,\ \delta = 15. \tag{2}$$

Moreover, $P_{\mathrm{RS}}(p) \to 1$ exponentially fast in n_1 once p exceeds the lift threshold $p_* = (n_1 - \delta)/n_1 = 31/46 \approx 0.674$: by Hoeffding's inequality,

$$P_{\mathrm{RS}}(p) \geq 1 - \exp\!\big(-2\, n_1 \, (p - p_*)^2\big) \quad \textit{for all } p > p_*. \tag{3}$$

Proof. The number of correctly recovered symbols follows Binomial(n_1, p); $\mathcal{D}_{\mathrm{RS}}$ succeeds iff this count is at least $n_1 - \delta$, giving (2). (3) follows from Hoeffding's tail bound applied to the empirical fraction of correct symbols.

Remark 1 (on the independence assumption). The i.i.d. assumption is idealized. In practice the per-byte CNN classifiers operating on adjacent RM blocks share trace context and their errors are mildly correlated. The empirical results of Section 3.5 nonetheless track (2) closely, indicating that the independence model is an adequate first-order abstraction for HQC's structured per-symbol POIs.

Two consequences follow immediately. First, the attacker's required per-byte accuracy collapses from the p^{16} law of raw recovery to a much softer binomial-tail criterion in Eq. (2), as Figure 3 makes concrete: at a per-byte accuracy of $p \approx 79\%$, direct 16-byte message recovery succeeds with probability only $P_{\mathrm{raw}} = 0.79^{16} \approx 2.3\%$, whereas passing the recovered 46-byte word through HQC's own RS decoder lifts this to $P_{\mathrm{RS}} \approx 97.8\%$. Second, and more fundamentally, the side-channel attacker's effective noise tolerance is *identical* to that of the legitimate decoder:

Corollary 1 (Side-Channel Robustness Equivalence). *Under the conditions of Theorem 1, the side-channel attacker succeeds whenever the symbol-error count $E(\hat{\mathbf{c}}, \mathbf{c}) \leq \delta$, which is exactly the same condition that determines decryption-failure-free operation of the legitimate receiver. Consequently, the reliability margin engineered into HQC by raising the RS correction radius simultaneously acts as a side-channel robustness margin in favor of the attacker.*

3.5 Experimental Validation on STM32F415

To validate Theorem 1 against real measurements, we evaluated the attack on STM32F415 in both the *raw* (no post-correction) and *RS-corrected* settings. We

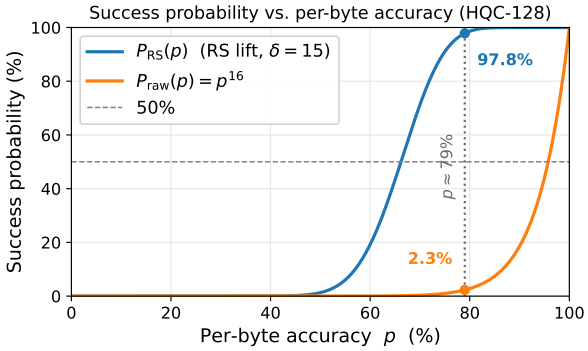


Fig. 3. Idealized single-trace success probability vs. per-byte accuracy p on HQC-128 (Theorem 1). Orange: direct full-message recovery $P_{\text{raw}}(p) = p^{16}$. Blue: RS post-corrected recovery $P_{\text{RS}}(p)$ from (2). The dashed line marks the $p \approx 79\%$ operating point discussed in the text.

captured power traces of the RM encoder, the 8-bit `encode` routine of Listing 1.2 run 46 times per C.Encode, once for each RS-codeword byte, and cut each trace at those 46 invocations. The routine is identical at every byte position, so the segments are pooled: one profiling trace supplies 46 training examples, and one classifier per bit is trained on all of them and then applied to every position at inference. The only difference between the two settings is whether the recovered 46-byte word is passed through the HQC RS decoder before declaring success.

Triggering and trace alignment. Every classifier in this paper consumes a window cut from a triggered region, and the cut is made per trace rather than at a fixed offset. The target firmware raises the ChipWhisperer trigger line immediately around the routine under study and lowers it immediately after; here that is each of the 46 `encode` calls, giving 46 pulses per trace. The trigger waveform is digitized alongside the power channel, and each trace's own rising edges are recovered from it at analysis time, so the 46 segments are cut at that trace's own edges. This matters: the interval between the serial command and the first edge drifts by 800–1300 samples across a capture, and cutting every trace at the offsets measured on the first one costs roughly a factor of two in $|t|$ at the leaking samples. Traces whose edge count differs from the expected one, one in 700,000 in the largest capture, are dropped rather than realigned.

Table 1 reports, as a function of the number of profiling traces, the mean per-bit and per-byte accuracy, the raw recovery rate before RS correction (*Full Msg.*, all 16 message bytes correct), the worst-case symbol-error count in the 46-byte estimate (*Err. max*, which the public RS decoder corrects whenever it is $\leq \delta = 15$), and the success rate after that decoder (*RS dec.*). All figures are computed over 1,000 evaluation traces disjoint from the profiling set: the accuracy and rate columns are means over that set, and *Err. max* is the maximum over it.

Table 1. RS post-correction on STM32F415 as a function of the profiling-trace budget. Highlighted: the first budget reaching 100% RS-decoded recovery.

Traces	Bit Acc. (%)	Byte Acc. (%)	Full Msg. (%)	Err. max	RS dec. (%)
20	90.86	45.61	0.00	39	0.4
40	96.56	75.86	1.27	22	92.8
60	97.52	82.23	4.56	18	99.5
80	97.96	85.18	**7.56**	15	**100.0**
100	98.33	87.66	11.98	14	100.0
200	98.90	91.78	26.12	11	100.0
300	99.17	93.82	36.21	10	100.0
400	99.35	95.07	44.50	9	100.0
500	99.42	95.66	49.89	7	100.0

The raw and RS-corrected columns differ sharply. At 60 profiling traces, raw recovery succeeds on 4.6 % of the evaluation set while RS decoding already succeeds on 99.5 %; from **80 traces onward, RS-corrected recovery reaches 100 %**, whereas raw recovery reaches only 49.9 % even at 500 traces, the largest budget tested. At 80 traces, the worst-case number of erroneous RS symbols across the evaluation set drops to exactly $\delta = 15$, the RS correction radius, which is precisely the threshold at which the public RS decoder can correct *every* recovery in the evaluation set, and from this point onward the RS-decoded recovery rate stays at 100%.

These measurements track the prediction of Theorem 1 closely: once the empirical per-byte accuracy crosses $p_* \approx 0.67$ (around 40 traces) and reaches the near-certainty regime by $p \approx 0.85$ (around 80 traces), the RS decoder absorbs every residual symbol error in the evaluation set.

3.6 Build Scope: Implementations and Optimization Levels

Section 3.5 characterizes a single build. We repeated the measurement on all three HQC codebases (NIST-ref, NIST-opt and PQClean), each compiled at `-Os` and `-O3` and flashed separately onto the same STM32F415 in an independent capture campaign, with 400 profiling and 400 attack traces per build. Table 2 reports the mean per-byte accuracy over the 46 codeword positions, with its spread across positions, the raw 16-byte message recovery rate before any RS correction (*Full Msg.*, as in Table 1), and the success rate after that decoder (*RS dec.*). The raw rate varies widely across builds, from 40.75% to 88.75%, but RS decoding absorbs the difference: all six reach 100%.

3.7 Discussion: The Reliability Margin as an Attack Budget

The RS code in HQC exists to drive down the decryption failure rate, and in legitimate decapsulation its $\delta = 15$ correction budget is almost never used: the inner RM decoder already removes essentially all errors, leaving the RS decoder's

Table 2. The RM-encoder attack across the three HQC codebases at `-Os` and `-O3`. †Identical to the HQC team's GitLab reference (`v5.0.0`, commit `d622142a`).

Implementation	Opt.	Byte Acc. (%)	Full Msg. (%)	RS dec. (%)
NIST-ref†	`-O3`	95.88 ± 0.92	48.25	**100.0**
	`-Os`	95.83 ± 1.44	49.75	**100.0**
NIST-opt	`-O3`	94.86 ± 0.99	40.75	**100.0**
	`-Os`	98.25 ± 0.70	72.75	**100.0**
PQClean	`-O3`	94.82 ± 1.18	45.75	**100.0**
	`-Os`	99.15 ± 0.56	88.75	**100.0**

input error-free except with the internal Reed–Muller code's decoding-failure rate: $2^{-10.96}$, $2^{-14.39}$, and $2^{-11.48}$ for HQC-128, HQC-192, and HQC-256 respectively [24]. A side-channel attacker, by contrast, exploits the full radius: each of the 15 correctable symbols is a slot in which a noisy per-byte estimate may be wrong yet still lifted to the exact codeword. The reliability margin thus simultaneously serves as an exploitable error budget for the side-channel adversary, which is the structural reason RS-aware single-trace recovery stays robust on noisy platforms (Theorem 1). We invoke the same lift again in Section 5, where it recovers the per-decapsulation mask $\mathbf{m}'$ single-trace and thereby helps strip the codeword-masking countermeasure meant to defend against this leakage.

4 $\mathbf{v} - \mathbf{u} \cdot \mathbf{y}$ Recovery: Necessity of Multi-POI Selection

Section 3 attacked the encoder side; we now turn to the complementary decoder-side venue of Maillet et al. [36], whose novel replay attack exposed the leakage of $\mathbf{v} - \mathbf{u} \cdot \mathbf{y}$ in the inner Reed–Muller decoder's `expand_and_sum` routine, recalled next in Section 4.1. Our finding is that their single-POI replay model, though sound, saturates well below per-bit certainty on our STM32F415 target when it is built at `-O3` rather than their `-O2`; combining several temporally-separated points of interest per target bit narrows the gap but does not close it.

4.1 Maillet et al.'s $\mathbf{v} - \mathbf{u} \cdot \mathbf{y}$ Attack

Maillet et al. [36] exploit the `expand_and_sum` routine of the inner Reed–Muller decoder. To compare directly with their analysis, we take the routine from NIST-opt rather than NIST-ref, reproduced verbatim in Listing 1.3. The routine takes the noisy duplicated codeword $\mathbf{v} - \mathbf{u} \cdot \mathbf{y}$ (viewed as d copies, MULTIPLICITY in the code, of a 128-bit RM word, each packed into eight `u16` halfwords) and produces a 128-element integer accumulator `dst->i16[0..127]` by summing all copies position by position. For HQC-128 the duplication factor is $d = 3$, so each accumulator position k is built up over three phases that we label `cp0`, `cp1`, `cp2`: `cp0` writes the single bit `stored`$[k] \in \{0, 1\}$, `cp1` adds `stored`$[k + 128]$ to

reach $\{0,1,2\}$, and cp2 adds stored$[k+256]$ to reach $\{0,1,2,3\}$. We index the 384 target bits as $b \in [0,384)$, with cp0, cp1, cp2 covering $[0,128)$, $[128,256)$, $[256,384)$ respectively, and use this labeling throughout Section 4. In the code, the structure factors into two phases: a *first-copy* loop (lines 3–7, cp0) that initializes the accumulator and a *rest-copies* loop (lines 9–15, cp1 and cp2) that updates it with += for the remaining copies.

```
1  void expand_and_sum(expandedCodeword *dst, codeword src[]) {
2      // first copy
3      for (size_t part = 0; part < 8; part++) {
4          for (size_t i = 0; i < 16; ++i) {
5              dst->i16[(part << 4) + i] = src->u16[part] >> i & 1;
6          }
7      }
8      // second, third copies
9      for (size_t copy = 1; copy < MULTIPLICITY; copy++) {
10         for (size_t part = 0; part < 8; part++) {
11             for (size_t i = 0; i < 16; ++i) {
12                 dst->i16[(part << 4) + i] += src[copy].u16[part] >> i & 1;
13             }
14         }
15     }
16 }
```

Listing 1.3. Reed–Muller expand_and_sum routine in NIST-opt (reproduced from [36]). In Maillet et al.'s model the first-copy leakage model carries over unchanged to the rest-copies loop, so a single model covers all copies.

The inner loop of either phase compiles, across the architectures tested in [36] (ARM T32, RISC-V, and x86-64 without SIMD), to four micro-architectural instructions: load, srl (right-shift), andi, and sh. A Hamming-distance leakage model is applied to the register transition induced by the srl, which exposes each bit of a target 16-bit block as a closed-form HD between successive register states. The attacker model assumes the ability to replay the same decryption on the target. With S replays of the same input, the sample mean at a fixed point of interest preserves the single-trace means μ_0, μ_1 but reduces the variance from σ^2 to σ^2/S, so that Bayes-optimal thresholding pushes the per-bit success probability $P_{\text{bit}}(S)$ toward 1 as S grows. A single trace gives $P_{\text{bit}} \in [0.6996, 0.7616]$ depending on the bit position; with $S = 83$ replays the per-bit probability is driven above 0.9998 and the full $\mathbf{v} - \mathbf{u} \cdot \mathbf{y}$ is recovered with success rate above 99% on a Cortex-M4 target. Knowing $\mathbf{v} - \mathbf{u} \cdot \mathbf{y}$ together with the public ciphertext $(\mathbf{u}, \mathbf{v})$ yields $\mathbf{u} \cdot \mathbf{y}$, from which the secret $\mathbf{y}$ is recovered [36]: when $\mathbf{u}$ is invertible in $\mathbb{F}_2[X]/(X^n - 1)$ the recovery is direct, and even in the non-invertible case at most a single bit remains to brute-force, so the attack amounts to a full key-recovery attack (see [36, §4] for both cases). The authors propose three countermeasures: redesigning the RM decoder as a minimum-distance decoder, adding noise to the ciphertext, and *codeword masking* in the sense of Merli et al. [39], which replaces $\mathbf{v} - \mathbf{u} \cdot \mathbf{y}$ with $\mathbf{v} - \mathbf{u} \cdot \mathbf{y} + \mathbf{m}'\mathbf{G}$ for a fresh random $\mathbf{m}'$ per decapsulation. Crucially, [36] confines its leakage analysis to the first-copy loop (lines 3–7) of Listing 1.3, and the closed-form HD derivation is presented for a single 16-bit block of one copy.

We build on this attack along two complementary directions. First, we extend the leakage analysis from the first-copy fragment (cp0) to the rest-copies loop

(lines 9–15, cp1 and cp2), where the same 16-bit block is accumulated again via +=. Maillet et al.'s Hamming-distance model on the srl register transition fully accounts for the cp0 leakage; on our STM32F415 target, we observe that cp1 and cp2 also leak, but at *different* points, dominated by the accumulator read–modify–write rather than the shift-register transition, which the single-POI HD model does not capture. Recovering this additional leakage therefore calls for a different, multi-POI analysis, which we develop in Section 4. Second, we revisit the codeword-masking countermeasure of [36] under a replay adversary that can additionally observe each per-decapsulation mask (Section 5).

4.2 Maillet's Leakage Hypothesis and What We Observe on F415

Maillet et al. [36] adopt the standard register-transition leakage model $L(X, Y) = \mathrm{HW}(X \oplus Y) + \mathcal{N}(0, \sigma)$ and localize the leakage of the NIST-opt RM decoder to a single instruction inside the expansion loop `dst[i] = (src[part] >> i) & 1`: the right-shift commitment on register `t1`. Each iteration overwrites `t1` with a right-shifted version of the previous content, and the HD between two consecutive `t1` states isolates one secret bit per iteration ([36, §5.1, Eq. (2)]). Under this hypothesis a single sample inside the expansion loop is informationally complete for each target bit, and Maillet's POI selection is a binary partition on the target bit value picking the sample with maximum class separation.

We re-examine the same primitive on STM32F415 (NIST-opt's u16 halfword `expand_and_sum` of Listing 1.3) and run the TVLA of Section 3.2, partitioned on `stored`$[b] = 0$ vs 1, for representative bits in each `cp` region. Here the capture is triggered around the single `expand_and_sum` call, and the analysis window is $[\max_i r_i, \min_i f_i]$ over the profiling set, the largest window lying inside the triggered region of *every* trace, so sample 0 means the same point in the execution for all of them. Figure 4 shows the result.

Two observations are immediate. First, the leakage on F415 is *not* concentrated at a single sample inside one loop body; each target bit produces multiple temporally-separated POIs spread across the three accumulator phases. Second, the number of POIs is structured: it equals $3 - \mathtt{cp}(b)$. A `cp` $= 0$ bit leaks at three positions (the cp= 0 accumulator write of `stored`$[k]$, plus the subsequent cp= 1 and cp= 2 reloads of `dst`$[k]$ that depend on it); a `cp` $= 1$ bit leaks twice (its own cp= 1 accumulator update plus the cp= 2 reload); a `cp` $= 2$ bit leaks once (its own cp= 2 update, with no further reload before the Hadamard transform).

The HD transition model that Maillet adopts continues to apply; what changes is *which* register write dominates, and on closer inspection this is a compilation effect rather than a device one. We build the RM decoder with `arm-none-eabi-gcc` 13.2.1 at `-O3`. At this level the compiler fully unrolls the inner bit loop and extracts each bit with a fixed-position `ubfx`, so no persistent shift register carries the per-iteration Hamming distance that Maillet's model localizes on (the variable shift `srl`/`asr`). This is purely a build choice: recompiling the same source at `-O2` reproduces Maillet's rolled-loop shift register (`asr`) exactly, on the same Cortex-M4. At `-O3` the dominant transition is instead the accumulator `dst`$[k]$, which is written at cp0 and then *reloaded* and rewritten by a read–modify–write

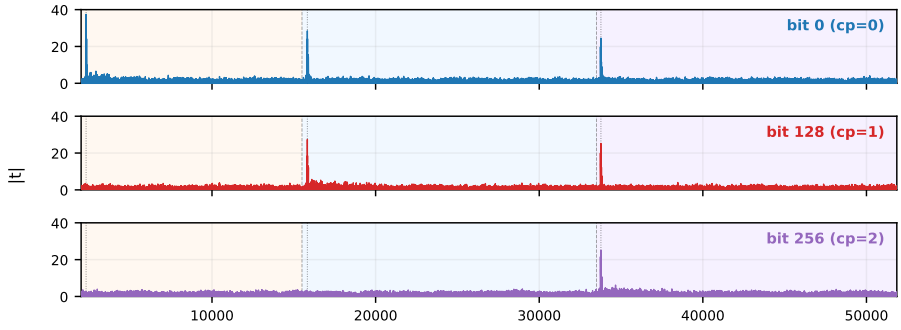


Fig. 4. TVLA of the u16 halfword `expand_and_sum` on STM32F415, partitioned on `stored`$[b] = 0$ vs 1, for $b = 0$, $b = 128$ and $b = 256$, the three source bits summed into the same accumulator position `dst`[0]; x is trimmed to the `expand_and_sum` window. The peak count equals $3 - \texttt{cp}(b)$: three for $b = 0$ (`cp`0), two for $b = 128$ (`cp`1), one for $b = 256$ (`cp`2). Dotted verticals mark the three writes of `dst`[0], where the surviving peaks coincide.

(`ldrh dst`$[k]$; `add`; `strh dst`$[k]$) at each later phase that incorporates a copy depending on it, exactly the $3 - \texttt{cp}(b)$ peak structure of Figure 4. This decreasing-with-`cp` trend is independently corroborated by Dong et al. [20], who attack the same `-O3` reference build on a Cortex-M4 and report per-block bit-recovery accuracies of 0.97, 0.81, and 0.72 for bit ranges 0–127, 128–255, and 256–383 (that is, `cp`0, `cp`1, `cp`2), the same monotone drop that Maillet's single-POI `-O2` model, under which one sample is informationally complete for every bit, does not exhibit. The same three bits appear in their TVLA [20, Fig. 6]. Because the leaking instruction is therefore not fixed across builds, a POI extractor that commits to the single shift instruction would miss the `cp`1 and `cp`2 reload leaks; we instead select points of interest *comprehensively*, without committing to a hypothesized instruction. Figure 5 places the two compilations side by side.

From 200,000 $\mathbf{v} - \mathbf{u} \cdot \mathbf{y}$ profiling traces on STM32F415 we compute, per target bit, the bit-conditioned t-test (the single-POI baseline) together with the sub-pair t-tests of [36] that condition on the two context copies at the same position, comparing the four sub-groups by both Hamming-distance and Hamming-weight contrasts. Their union captures multi-phase leakage without committing to which instruction is responsible; the top-$K = 3$ peaks per bit feed the multi-POI strategy of the next subsection.

4.3 Single-POI Saturation and The Multi-POI Lift

Averaging R same-input replays at one POI cuts the noise from σ^2 to σ^2/R, so under Maillet's hypothesis the per-bit rate tends to 1, provided the leakage sits at one instruction. On F415 it does not (Figure 4), so keeping only the largest peak should leave information behind whatever R is. We test this with two independent single-POI baselines:

Maillet et al., ARM T32, `-O2`

```
ldrh.w r7,[r6,#2]! @ load src
asr.w  r4,r7,r3    @ shift -> r4
and.w  r4,r4,#1    @ extract bit
strh.w r4,[r5],#2  @ store bit
```

HD on the shift register `r4` (rolled loop).

This work, ARM T32 (F415), `-O3`

```
; first copy (cp0)
ldrh.w r4,[r2,#2]! @ load src
ubfx   r4,r4,#i,#1 @ fixed extract
strh.w r4,[r3,#-k] @ write dst[k]
; rest copies (cp1,cp2)
ldrh.w r1,[r2,#2]! @ load src
ldrh   r4,[r3]     @ reload dst[k]
add    r1,r4       @ accumulate
strh   r1,[r3]     @ write dst[k]
```

HD on the accumulator `dst[k]`
(unrolled `ubfx`, no shift register).

Fig. 5. Inner body of `expand_and_sum` on the same ARM T32 (Cortex-M4) core: `-O2` (left; from Maillet et al. [36], Fig. 4) vs `-O3` (right; this work). `-O2` keeps each bit in a shift register (`asr`); `-O3` unrolls to `ubfx` and the dominant leakage moves to the accumulator `dst`$[k]$.

– [36] – its §6 Gaussian template, profiled per bit at the bit-conditioned POI from 100,000 traces and decided by the Bayes LLR under $\mathcal{N}(\mu_b, \sigma_b^2/R)$ on the mean of the R POI values.
– single-POI – a per-bit 1D CNN on the 128-sample window at the same POI (20,000 profiling traces, 8 epochs), with the R replay softmaxes summed at inference.

They aggregate the replays differently: the template *averages* the raw POI value, so replays provide only noise suppression, while the CNN *soft-votes* over per-trace posteriors. The distinction looks minor here but is decisive in Section 5, where only soft voting can fold each replay's mask into the decision.

Both baselines are evaluated on the same dataset: the 1,000 fixed v_{uy} ciphertexts, each replayed R times with masking disabled, are split 500/500 into a validation and a held-out test set, and the CNN accuracies in Figure 6 are reported on the 500-ciphertext test set; the single-POI curves are the relevant ones here.

The cp= 0 block saturates at 1.0 already at $R = 1$: the accumulator initialization `dst`$[k]$ = `stored`$[k]$ is an informationally complete first POI, so a single sample suffices. On cp= 1 and cp= 2, both single-POI baselines saturate well below 1 and barely move with R (Figure 6). Maillet's cp= 2 accuracy is even non-monotone in R ($0.5965 \to 0.5769$): the POI-value distribution there is a mixture over the accumulator values $\{0, 1, 2, 3\}$ rather than a single Gaussian, so averaging more replays only pulls $\bar{x}$ toward a mean the template never modeled.

The single-POI ceiling is therefore not a methodological artifact of any one classifier but structural to the venue; we pin down its exact cause in Section 4.4. Extracting the remaining information requires combining POIs from different phases, exactly the multi-peak structure of Figure 4, which is what the multi-POI strategy does next.

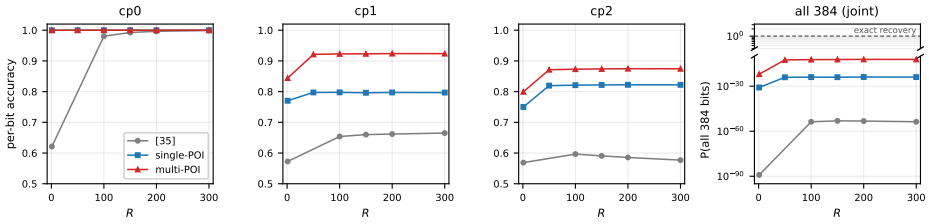


Fig. 6. Per-bit accuracy of the three replay strategies ([36], and single- and multi-POI CNNs) on $\mathbf{v} - \mathbf{u} \cdot \mathbf{y}$ (STM32F415, unmasked mode) against replay count R, on the 500-ciphertext held-out test set. The first three panels report the *mean* of the per-bit accuracies over the 128 bits of that `cp` region. The fourth reports a different quantity: the *product* of all 384 per-bit accuracies, that is the probability of recovering the entire 384-bit $\mathbf{v} - \mathbf{u} \cdot \mathbf{y}$ block with every bit correct, plotted on a logarithmic axis; bits are treated as independent. Strategies are defined in the text.

Multi-POI stacks the top-$K = 3$ peaks of Section 4.2 into a $\mathbb{R}^{3\times128}$ feature for the same per-bit CNN. Peaks are admitted from $\texttt{cp} \in \{0, 1\}$ for a `cp1` bit and $\texttt{cp} \in \{1, 2\}$ for a `cp2` bit; excluding `cp0` from the latter is an empirical choice.

At $R = 300$ multi-POI reaches 0.9237 on `cp1` and 0.8744 on `cp2`, the best of the three strategies in every region and at every R; essentially all of the movement is over by $R = 100$. Two caveats: the ceiling is specific to the `-O3` accumulator leak (under Maillet's `-O2` build it does not arise), and multi-POI still reaches ≈ 1.0 on `cp0`, so it generalizes the model of Maillet et al. rather than replacing it.

4.4 The Accumulation Ceiling

Why does a residual gap survive even multi-POI soft voting? Because each accumulator position is built up by $\texttt{dst}[k]$ += $\texttt{stored}[k + 128c]$ over the phases $c = 0, 1, 2$, the value exposed at any single phase packs together *every* source bit summed so far. Figure 4 already shows this for $k = 0$: the single peak of $b = 256$ sits at the same sample as the last peak of $b = 0$ and of $b = 128$. That sample is the `cp2` write of $\texttt{dst}[0]$, where the leak carries $\texttt{stored}[0] + \texttt{stored}[128] + \texttt{stored}[256]$, whereas the `cp1` write carries $\texttt{stored}[0] + \texttt{stored}[128]$ and the `cp0` write only $\texttt{stored}[0]$: three secret bits collapsing into a single observed value. No soft vote over *identical-context* replays can separate them: every replay re-draws the same three-bit sum, so the ambiguity is irreducible. This is the structural cause of the `cp1`/`cp2` ceiling, and multi-POI only mitigates it (by recovering the cleaner earlier-phase leaks), never removes it.

The per-bit figures of Figure 6 are *means*, and a mean is not a recovery probability. Recovering one `expand_and_sum` block means getting all 384 of its bits right at once, so the per-bit accuracies compound into a product across the block: multi-POI's mean of 0.9327 at $R = 300$ yields 7.1×10^{-13}, and Maillet's 0.747 yields $10^{-53.7}$. A full $\mathbf{v} - \mathbf{u} \cdot \mathbf{y}$ consists of 46 such blocks, which must all succeed together, so each block figure is raised to the 46th power: $(7.1 \times$

$10^{-13})^{46} \approx 10^{-559}$ for multi-POI, against $(10^{-53.7})^{46} \approx 10^{-2470}$ for Maillet's single-POI attack. Here and throughout, one 384-bit block is what we measure; every full-vector figure in this paper is that measurement raised to the 46th power under bit independence, not a 17,664-bit recovery we ran. Both figures remain far beyond reach, and the requirement is exact: $\mathbf{y}$ is recoverable from $\mathbf{v} - \mathbf{u} \cdot \mathbf{y}$ only when that vector is known without error (Section 4.1), so a single erroneous bit leaves the key unrecovered.

The consequence is that even this raised ceiling leaves exact recovery out of reach. Multi-POI lifts the per-bit accuracy far above Maillet's 0.747, and the block-level probability with it, yet exact recovery of even a single block, let alone the whole $\mathbf{v} - \mathbf{u} \cdot \mathbf{y}$ and hence the key, remains infeasible from identical replays alone. The hybrid multi-POI strategy of this section therefore establishes a methodological floor on the `expand_and_sum` venue rather than closing the gap to exact recovery on its own. Whether that gap can be closed depends not on the attack algorithm but on the deployment configuration: specifically, on whether the receiver applies a randomized codeword mask between successive decapsulations. Section 5 analyzes that configuration.

5 Codeword Masking's Amplification of $\mathbf{v} - \mathbf{u} \cdot \mathbf{y}$ Recovery

Section 4 established a methodological floor of ≈ 0.93 per-bit accuracy, a mean taken over the 384 bits of the block, on unmasked-mode $\mathbf{v} - \mathbf{u} \cdot \mathbf{y}$ recovery: the multi-POI hybrid strategy escapes the single-POI ceiling but leaves a residual per-bit error of ≈ 0.067 that prevents bit-exact recovery of the 17,664-bit codeword. Closing the remaining gap requires information beyond what the attack algorithm itself can extract from identical replays.

This section shows that one of the countermeasures proposed by Maillet et al. [36, §7.3], codeword masking, supplies exactly that information when the attacker can recover the per-decapsulation mask. The very randomness that the countermeasure introduces to hide $\mathbf{v} - \mathbf{u} \cdot \mathbf{y}$ becomes, under a known-mask adversary, the input diversity that breaks the single-POI accumulator ambiguity. We formalize this effect, instantiate the known-mask adversary using the single-trace RM-encoder leakage of Section 3, and report that at $R = 500$ the probability of recovering all 384 bits of the block rises from $10^{-12.1}$ (unmasked-mode hybrid) to 1.0000 (masked-mode hybrid), and over the full 17,664-bit $\mathbf{v} - \mathbf{u} \cdot \mathbf{y}$ from $10^{-554.9}$ to 1.0000 (Table 3): Maillet et al.'s replay key-recovery attack, which the unmasked-mode ceiling left short of bit-exact $\mathbf{v} - \mathbf{u} \cdot \mathbf{y}$ recovery, becomes *stronger* once codeword masking is deployed. The countermeasure amplifies the very attack it was intended to thwart.

5.1 Maillet's Codeword Masking Proposal

Maillet et al. [36, §7.3] adopt the codeword-masking primitive of Merli, Stumpf and Sigl [39] as a software countermeasure against the very $\mathbf{v} - \mathbf{u} \cdot \mathbf{y}$ leakage we exploit. The receiver draws a fresh random message $\mathbf{m}'$ per decapsulation,

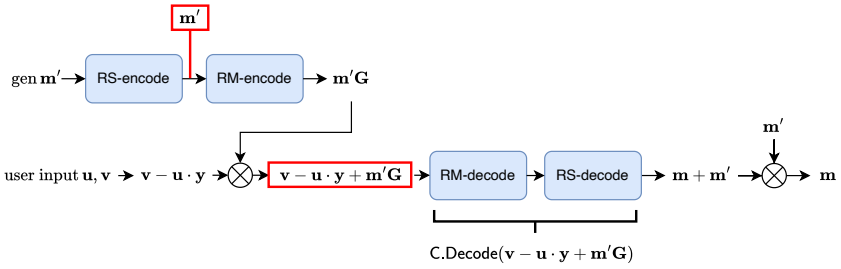


Fig. 7. Codeword-masking countermeasure of [36, §7.3]: a fresh $\mathbf{m}'$ is encoded and added on the ciphertext side, so `expand_and_sum` sees the randomized $(\mathbf{v}-\mathbf{u}\cdot\mathbf{y})+\mathbf{m}'\mathbf{G}$; $\mathbf{m}'$ is subtracted from the decoder output to unmask.

encodes it through the same RM/RS pipeline as the legitimate message ($\mathbf{m}'\mathbf{G}$), and adds it to the ciphertext side before decoding:

$$\mathbf{v}' \;=\; \mathbf{v} \;+\; \mathbf{m}'\mathbf{G}. \tag{4}$$

The decoder then computes $\mathsf{C.Decode}(\mathbf{v}' - \mathbf{u} \cdot \mathbf{y}) = \mathbf{m} + \mathbf{m}'$, and the unmasking step recovers $\mathbf{m}$ by subtracting $\mathbf{m}'$. The `expand_and_sum` routine therefore now operates on $\mathbf{v}' - \mathbf{u} \cdot \mathbf{y} = (\mathbf{v} - \mathbf{u} \cdot \mathbf{y}) + \mathbf{m}'\mathbf{G}$ rather than on $\mathbf{v} - \mathbf{u} \cdot \mathbf{y}$ alone, randomizing the register state that previous attacks exploited. Figure 7 sketches this pipeline.

Under the assumption that $\mathbf{m}'$ remains secret to the attacker, the countermeasure is sound: each replay of the same ciphertext now has a fresh random offset on the register state, so the per-trace POI value distribution conditioned on $(\mathbf{v} - \mathbf{u} \cdot \mathbf{y})[b]$ becomes a marginal over the random mask and the conditional bit information vanishes in the marginal. Maillet et al. confirm this experimentally on their RISC-V simulator: a t-test on 30 samples detects no remaining leakage, and the runtime overhead is $+1.14\%$ [36, Table 5]. We do not dispute the countermeasure under its stated threat model. The result of this section is that the threat model breaks the moment the attacker can observe each $\mathbf{m}'$.

5.2 Vulnerability Amplification Under Known-Mask Adversaries

We now drop the assumption that $\mathbf{m}'$ stays secret. Suppose the attacker observes each per-decapsulation $\mathbf{m}'_r$ for the R same-ciphertext replays of the attack ($r = 1, \ldots, R$). Section 5.3 will show that this assumption is realized in practice on STM32F415; the present subsection treats it as a hypothesis and derives the attacker's resulting posterior on $\mathbf{v} - \mathbf{u} \cdot \mathbf{y}$.

Setup. Write the secret target as $\boldsymbol{\sigma} = \mathbf{v} - \mathbf{u} \cdot \mathbf{y}$ and let $g_{b,r} = (\mathbf{m}'_r\mathbf{G})_b \in \{0, 1\}$ denote the b-th component of the encoded mask for replay r. The bit that the

CPU writes into the `expand_and_sum` accumulator at position b during replay r is then

$$\texttt{stored}_r[b] \;=\; \sigma_b \oplus g_{b,r}. \tag{5}$$

The single-trace classifier (CNN or otherwise) of Section 4 estimates, from trace t_r, the per-trace posterior $q(s \mid t_r) := \Pr[\,\texttt{stored}_r[b] = s \mid t_r\,]$.

Bayes posterior on σ_b across R replays. Assuming the replays are conditionally independent given $(\sigma_b, \mathbf{m}'_{1..R})$ (which holds because the measurement noise and the random mask are independent across replays), the joint posterior factorizes:

$$\Pr[\sigma_b = v \mid t_{1..R}, g_{b,1..R}] \;\propto\; \prod_{r=1}^{R} q\big(v \oplus g_{b,r} \,\big|\, t_r\big)\,. \tag{6}$$

Equivalently, with $L_r := \log[\,q(1 \mid t_r)/q(0 \mid t_r)\,]$ denoting the per-trace log-likelihood ratio, the log-odds for σ_b are

$$\log \frac{\Pr[\sigma_b = 1 \mid \cdot]}{\Pr[\sigma_b = 0 \mid \cdot]} \;=\; \sum_{r=1}^{R} (-1)^{g_{b,r}}\, L_r. \tag{7}$$

Each replay contributes the per-trace log-LR L_r, sign-flipped according to the known mask bit $g_{b,r}$. The decision rule is $\hat{\sigma}_b = 1$ iff the sum in (7) is positive.

Why this escapes the single-POI ceiling. The single-POI ceiling of Section 4.4 is, in this language, a property of the accumulator value $a_r = \sum_k \texttt{stored}_r[k_{\mathrm{ctx}}]$ that the POI exposes: its conditional law $\Pr[a_r \mid \sigma_b = v]$ is a 2-mixture (cp= 1) or 3-mixture (cp= 2) over the unknown context bits, and the $v = 0$ and $v = 1$ mixtures share an interior value (e.g. $a_r = 1$) that caps the per-trace $|L_r|$. In unmasked mode every replay re-samples the *same* context, so this ambiguity recurs identically and the ceiling persists for any R.

When $\mathbf{m}'_r$ is fresh per replay, the context bits $\texttt{stored}_r[k_{\mathrm{ctx}}] = \sigma_{k_{\mathrm{ctx}}} \oplus g_{k_{\mathrm{ctx}},r}$ vary across r even though σ_b does not. A given replay's accumulator a_r thus draws from a different point of the same 2-mixture (resp. 3-mixture), so the residual ambiguity is sampled at *different context realizations* across replays. Combining R such draws marginalizes the context out of the posterior on σ_b, and the marginalized posterior is strictly more concentrated than the per-trace one. Concretely, if $g_{b,r}$ is uniform on $\{0, 1\}$ across replays, half the terms in (7) cast a positive vote and half cast a negative one if σ_b has the "wrong" value, but the context-conditioned magnitudes differ across replays exactly because $g_{k_{\mathrm{ctx}},r}$ varies; the sum therefore exhibits a non-trivial drift toward the correct sign as $R \to \infty$, in contrast to the constant-context unmasked-mode case where the drift caps at the ceiling. We give the empirical confirmation in Section 5.3.

5.3 Realizing The Known-Mask Adversary; Empirical Amplification

The argument of Section 5.2 hinged on the attacker observing $\mathbf{m}'_r$ for each replay. We now show that this is achievable on STM32F415, and report the resulting accuracy.

Recovering $\mathbf{m}'_r$ single-trace via the RM encoder. Section 3 established that the RM encoder inside FO re-encryption is a structured per-bit leakage venue: every message byte is RM-encoded once per decapsulation through the `sbfx`-based `BITOMASK` expansion pattern, and a profiled per-byte classifier together with the RS post-decoding lift of Theorem 1 recovers the entire encoded message bit-exactly from a *single* trace once ∼80 profiling traces are seen (Table 1, 100% RS-corrected recovery at 80 traces onward). The same machinery applies verbatim to $\mathbf{m}'$: a randomized codeword mask is, from the encoder's point of view, indistinguishable from any other 16-byte message that the FO re-encryption would encode. Concretely, during each masked decapsulation the receiver invokes the RM encoder on $\mathbf{m}'_r$ to produce $\mathbf{m}'_r\mathbf{G}$, and the single trace of that encoder execution leaks $\mathbf{m}'_r$ in the same way that any other message would leak. The attacker thus obtains $g_{b,r} = (\mathbf{m}'_r\mathbf{G})_b$ per replay, exactly the side information required by (7).

A masked capture carries both venues in one trace: 46 encode pulses followed by one `expand_and_sum` pulse ($3 \times 46 + 1$ for the three-share variant), so a single trace holds the mask encoding and the accumulation it feeds.

Inference: feed $g_{b,r}$ into the hybrid CNN. Operationally, we use the same per-bit *hybrid* strategy as Section 4.3: both a single-POI and a multi-POI CNN are trained on mask-randomized TRAIN traces, and the one scoring higher on the held-out validation ciphertexts is kept. Multi-POI is not uniformly better, since its wider input is easier to overfit, so the choice is made per bit. The masked-mode evaluation then runs as follows. For each masked-mode replay r:

(i) Extract the trace window at the bit's selected POI or POIs and pass it through the CNN to obtain the per-trace softmax $\big(q(0 \mid t_r),\, q(1 \mid t_r)\big)$ over the *stored* bit $\texttt{stored}_r[b] = \sigma_b \oplus g_{b,r}$.
(ii) Apply the known-mask XOR correction $q^\sigma(v \mid t_r) := q(v \oplus g_{b,r} \mid t_r)$ to convert to a per-trace posterior on σ_b.
(iii) Aggregate across R replays by summing the corrected softmaxes (Eq. (7) in the softmax-sum approximation): $\hat{\sigma}_b = \arg\max_v \sum_r q^\sigma(v \mid t_r)$.

Step (ii) is the only difference from the unmasked-mode pipeline of Section 4.3; concretely it amounts to flipping the two softmax components when $g_{b,r} = 1$ and leaving them unchanged when $g_{b,r} = 0$. Summing the softmax posteriors in step (iii) is the *soft-vote* approximation of the exact log-odds rule (7); on our calibrated CNN the two agree to within 0.2 pp, so we report the softmax-sum throughout.

Why soft voting, not signal averaging. Maillet's framework averages the R replay POI values ($\bar{x} = R^{-1} \sum_r x_r$) before a single Bayes classification, which is correct for *unmasked* replays, where every replay writes the same σ_b and only measurement noise varies, so $\bar{x}$ denoises toward the class mean. Under codeword masking the CPU instead writes $\texttt{stored}_r[b] = \sigma_b \oplus g_{b,r}$, so averaging $x_1, \ldots, x_R$ mixes *two different conditional distributions* (the $\sigma_b \oplus g_{b,r} = 0$ branch on roughly half the replays and the $= 1$ branch on the other), and $\bar{x}$ concentrates at the mid-point

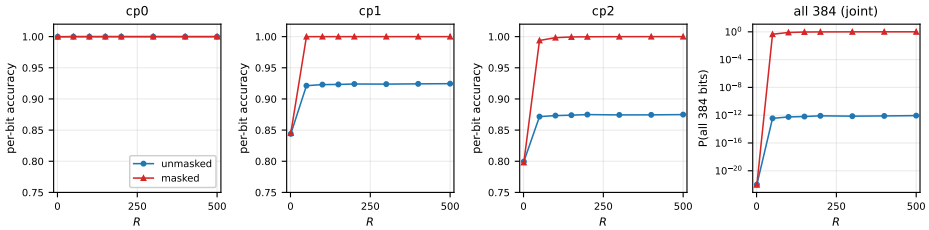


Fig. 8. End-to-end masked replay attack with the known-mask XOR correction (STM32F415), unmasked vs masked mode, against replay count R. The first three panels average the hybrid-POI per-bit accuracy over the 128 bits of the `cp` region; the fourth is different in kind, giving the probability that *all* 384 bits are recovered as the product of those accuracies under an independence assumption (log axis).

of the two class means, collapsing the Bayes decision to chance however large R becomes. The soft voting rule of steps (i)–(iii) sidesteps this by classifying each replay individually and applying the XOR correction at the per-trace stage (step (ii)), so that what is summed is consistent across replays. The diversity of $g_{b,\cdot}$ that averaging destroys is precisely what marginalizes the context bits (Section 5.2).

Empirical result. Figure 8 reports the resulting per-cp accuracy of the hybrid-POI strategy on the MASKED dataset (per-replay fresh $\mathbf{m}'_r$), against the same strategy on the unmasked dataset. Both use 200,000 profiling traces and the same 1,000 fixed v_{uy} ciphertexts, split 500/500 into a validation and a held-out test set, with every figure reported on the 500-ciphertext test set. That contrast is the central empirical claim of this section.

Table 3 gives the exact recovery probabilities. In unmasked mode, increasing the number of replays yields an initial improvement that rapidly plateaus; under masking, where soft voting treats each fresh mask as an independent sample, block recovery reaches 1.0000. Recovering the full vector imposes a much more stringent requirement: 0.8062 per block is still only $10^{-4.3}$ over 46 of them, and only at $R = 500$ does the attack become practical.

Summary. The codeword-masking countermeasure of [36, §7.3] is not a defense against an attacker who can observe the per-decapsulation mask: the RM-encoder venue of Section 3 supplies that observability single-trace, and the random masks then become the cross-replay diversity that breaks the single-POI accumulator ambiguity. Combined, the two effects move exact recovery of the full 17,664-bit $\mathbf{v} - \mathbf{u} \cdot \mathbf{y}$ at $R = 500$ on STM32F415 from $10^{-554.9}$ unmasked to 1.0000 masked. Contrary to the countermeasure's stated intent, it is masking that makes the attack practically viable.

Third-order masking. Splitting the mask into three shares does not mitigate the attack. The encoder runs once per share and all three encodings are added to

Table 3. Probability of exact recovery under the hybrid-POI attack against replay count R, for one 384-bit `expand_and_sum` block, and for the full 17,664-bit $\mathbf{v} - \mathbf{u} \cdot \mathbf{y}$ extrapolated from it as in Section 4.4. The 3rd-order rows are a separate capture of the same trace budget.

Mode	Target	$R = 1$	$R = 50$	$R = 100$	$R = 200$	$R = 500$
unmasked	one block	$10^{-22.0}$	$10^{-12.5}$	$10^{-12.3}$	$10^{-12.1}$	$10^{-12.1}$
	full $\mathbf{v} - \mathbf{u} \cdot \mathbf{y}$	$10^{-1011.6}$	$10^{-572.9}$	$10^{-563.6}$	$10^{-556.2}$	$10^{-554.9}$
masked 1st-order	one block	$10^{-22.0}$	0.4474	0.8062	0.9684	1.0000
	full $\mathbf{v} - \mathbf{u} \cdot \mathbf{y}$	$10^{-1012.7}$	$10^{-16.1}$	$10^{-4.3}$	0.2283	1.0000
masked 3rd-order	one block	$10^{-24.1}$	0.3440	0.7979	0.9743	1.0000
	full $\mathbf{v} - \mathbf{u} \cdot \mathbf{y}$	$10^{-1109.0}$	$10^{-21.3}$	$10^{-4.5}$	0.3014	1.0000

the ciphertext, so the bit the CPU writes is

$$\texttt{stored}_r[b] \;=\; \sigma_b \oplus g^{(1)}_{b,r} \oplus g^{(2)}_{b,r} \oplus g^{(3)}_{b,r}. \tag{8}$$

Step (ii) corrects with the XOR of the three mask bits rather than one, which is why the attacker needs every share. On the same budget (200,000 profiling traces and 1,000 replayed v_{uy} ciphertexts split 500/500 into validation and held-out test), the 3rd-order rows of Table 3 track the 1st-order ones at every replay count and reach 1.0000 at $R = 500$, exactly as the first-order case does. Recovering $\mathbf{m}'$ now means recovering all three shares, but that costs nothing extra to profile. Every share passes through the same RM-encoder code path, so a profile trained on $\mathbf{m}'_1$ from 200 traces (pooling their 46 encode segments) transfers unchanged to $\mathbf{m}'_2$ and $\mathbf{m}'_3$; all three reach 99.4% byte accuracy, RS-decode to the exact share, and are recovered simultaneously on 2000/2000 test traces. Raising the masking order increases the attacker's computational complexity during inference linearly, while leaving the required profiling complexity unaffected.

HQC-192 and HQC-256. Everything above is measured on HQC-128, whose duplicated Reed–Muller code repeats each codeword $d = 3$ times inside a 384-bit block. The two higher levels use $d = 5$ over a 640-bit block, so `expand_and_sum` folds five copies into each accumulator position and the block splits into `cp0` through `cp4`: one echo per copy, as in Figure 4, and an accumulation ceiling (Section 4.4) deeper than at $d = 3$, since a leaked accumulator value can now pack five secret bits. A bit-conditioned t-test confirms that the leak weakens with each further copy; Appendix D reports it. We repeated the end-to-end experiment there, and the amplification reproduces. The codeword is $n_1 = 56$ blocks for HQC-192 (35,840 bits) and 90 for HQC-256 (57,600), so exact recovery of the full vector at $R = 500$ stands at 10^{-843} and 10^{-1354} masked, against 10^{-4299} and 10^{-6909} unmasked. Masking raises the recovery probability, but even at $R = 1000$ it does not reach a practical value.

Figure 9 shows where the two modes part. The first three panels give the hybrid-POI per-bit accuracy at each accumulation step, with `cp1`, `cp2` and `cp3`, `cp4`

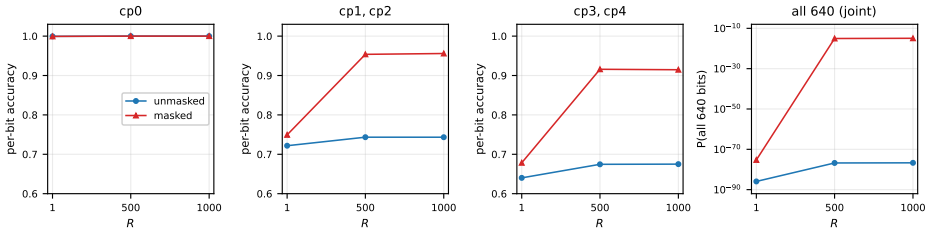


Fig. 9. The replay attack on HQC-192 ($d = 5$), unmasked vs masked mode, against replay count R. The first three panels average the hybrid-POI per-bit accuracy over the 128 bits of each cp region, with cp1, cp2 and cp3, cp4 pooled; the fourth gives the probability that all 640 bits of one block are recovered, as the product of those accuracies under an independence assumption (log axis).

pooled; cp0 is a plain assignment and is recovered outright in both modes, and from cp1 onwards the accumulator carries a sum of secret bits. At $R = 1$ the two modes track each other, since a single trace cannot tell them apart. Replay then separates them: the masked mode rises above 0.91 at every accumulation step while the unmasked one never passes 0.76, exactly the sign flip of Section 6.3 at $d = 3$. The fourth panel is where the deeper ceiling shows. At $R = 500$ one 640-bit block is recovered with probability $10^{-15.0}$ masked against $10^{-76.8}$ unmasked: masking is worth sixty orders of magnitude, the same direction it is worth at $d = 3$, but from a starting point five-fold accumulation has pushed far enough down that raising the block figure to the n_1-th power leaves the full vector out of reach at both higher security levels. Although the amplification effect remains observable at $d = 5$, it is insufficient for practical key recovery.

6 Discussion

6.1 The Significance of Message Recovery

Section 3 recovers the message from a single decapsulation, from about 80 profiling traces, and under the Fujisaki–Okamoto transform that message *determines* the shared key: decapsulation derives $(K', \theta') \leftarrow G\big(H(\mathsf{pk}_{\mathrm{KEM}}) \,\|\, \mathbf{m}' \,\|\, \mathsf{salt}\big)$ (Algorithm 2), so for an accepted ciphertext the recovered $\mathbf{m}$ directly determines K without requiring reconstruction of the long-term secret key. The session key is drawn afresh for each decapsulation, so averaging over repeated executions is unavailable and any attack on it must succeed from a *single* trace, the same target and the same requirement as Goy et al. [28], and the RM-encoder venue meets it. Recovering $\mathbf{m}'$ also directly resolves, without additional overhead, the plaintext-checking query that the PCA family of long-term key-recovery attacks presupposes; Appendix B develops that use and confirms it end-to-end.

Table 4. Adversary assumptions for the three attacks on $\mathbf{v} - \mathbf{u} \cdot \mathbf{y}$, all on **HQC-128**, `-O3`, Cortex-M4. R is the number of replayed decapsulations of one ciphertext.

Work	Type	Ciphertext	Unmasked	Order-t masked
[32]	non-profiled	chosen	50	3,000 at t=1; $t \geq 2$ open
[20]	profiled	randomness known	1	1 at t=1; $t \geq 2$ open
This work	profiled	**public only**	infeasible	$R \approx 500$**, any** t ($t \leq 3$ measured)

6.2 Comparison with Related Works

We compare against the two most recent attacks on this venue, Dong et al. [20] and Huang et al. [32]; Table 4 sets out the adversary each of them assumes.

Dong et al. attack the same venue we do, the masked $\mathbf{v} - \mathbf{u} \cdot \mathbf{y}$ handed to `expand_and_sum`, by a different route. The two works differ on two axes.

The first is the $\mathbf{v} - \mathbf{u} \cdot \mathbf{y}$ attack itself. Theirs is novel and far more efficient: a single trace recovers the long-term key, against $R \approx 500$ replays here. This higher efficiency comes at the expense of assuming a stronger adversary model. Both of their variants reduce key recovery to decoding $\mathbf{e}' = \mathbf{e} + r_1\mathbf{y} + r_2\mathbf{x}$ with $\mathbf{e}, r_1, r_2$ *known* to the attacker, either because the adversary generated the ciphertext or, in the active variant, because it chose them. We assume neither, and follow Maillet et al.'s model of honest decapsulations of a fixed ciphertext known only through its public $(\mathbf{u}, \mathbf{v})$. We do not claim to beat their efficiency on this venue; what we show instead is that masking helps the attack rather than hindering it.

The second is how the mask itself is recovered. Dong et al. strip $\mathbf{m}'$ through the Reed–Solomon encoder's `gf_mul`, using the operand-value leakage identified by Goy et al. [28], which is highly effective on the reference implementation they target. That venue exhibits substantial leakage for reasons analogous to our observations: the encoder's LFSR structure feeds each gate value to `gf_mul` `PARAM_G` $= 31$ times within a single execution, so an attacker can vote over those repetitions much as the post-decoding lift votes over RS symbols. Redundancy introduced for reliability is again what exposes the component. It is, however, sensitive to how the multiplication is written: swapping the commutative operands leaves Hamming-weight leakage, which a SASCA in the style of [28] could still exploit, and the faster RS encoder of Chen et al. [15] removes the value leakage altogether; Appendix E measures both on F415. We recover $\mathbf{m}'$ from the Reed–Muller encoder instead, lifted by RS post-decoding (Section 3); the two routes are complementary, and ours stays available where the multiplier has been rewritten. Dong et al. close by pointing to higher-order masking as the way forward; for codeword masking at least, raising the order does not raise the cost of recovering the mask, as we confirm up to $t = 3$ (Section 5).

Huang et al. mount a non-profiled DPA on `vect_add` with chosen ciphertexts, and also break the masked implementation, but at a cost that grows with the masking order: 50 traces unmasked become 3,000 at first order, and $t \geq 2$ is left open (Table 4). Our cost does not scale that way: recovering the mask does not

Table 5. Whether the attack succeeds, with and without codeword masking. Row 2 is measured here; rows 1 and 3 are as stated in the original papers. Rows 1–2 target `expand_and_sum` and differ in the build and the target board; row 3 targets `vect_add`. HQC-128 throughout. The last column qualifies the masked entry.

Work	Venue	Opt.	Unmasked	Masked	Note
[36]	`expand_and_sum`	`-O2`	✓	✗	TVLA only, not attacked
This work	`expand_and_sum`	`-O3`	✗	✓	measured in this work
[32]	`vect_add`	`-O3`	✓	✓	at 60× the traces

get more expensive to profile with the order, since one profile transfers to all three shares of a $t = 3$ implementation (Section 5).

6.3 What This Means for Deploying Codeword Masking

Table 5 sets our result beside the attack that motivated the countermeasure in the first place. Maillet et al. [36] recover $\mathbf{v} - \mathbf{u} \cdot \mathbf{y}$ on an STM32F303 at `-O2` and propose codeword masking to stop it, supporting the proposal with a t-test that finds no leakage, although on a simulator, with 30 samples, rather than on the target they attacked. We revisited both the attack and the countermeasure on an STM32F415 at `-O3`, and observe an inversion in countermeasure efficacy. The unmasked attack becomes ineffective: the compiler keeps the accumulator in a register across all three copies, and even after rebuilding the POI selection around that code shape the multi-POI strategy of Section 4 stays far short of recovery, whereas the application of codeword masking renders the attack viable.

This deviates from the typical failure mode of cryptographic countermeasures. Conventionally, masking increases the attacker's required trace budget, as in the third row of Table 5, where the unmasked attack succeeds and masking leaves the implementer a real if finite margin. In this case, however, masking does not increase attack complexity, but instead supplies the per-replay diversity that the accumulator ambiguity needs in order to be marginalized away. The condition is that the shares be recoverable, which for this construction they are.

The same lesson has been reaching constant-time coding from the other side. `KyberSlash` [6] begins where a secret-dependent division leaks through execution time; the defense is to write the code so that it runs in constant time, and the reference implementation does. What Bernstein et al. show is that the defense survives only some builds: asking `gcc` to optimize for code size (`-Os`) disables the conversion of the division into a multiplication, and the timing channel returns. Dong and Guo [19] find the same failure one level up. HQC's AVX2 implementation selects on secret data through a branch-free mask, which is constant-time as written, but `clang-14` and `clang-15` at `-O3` rewrite that mask into a chain of conditional branches, and the secret reaches the cache. In both cases the countermeasure is present in the source and absent from the binary.

Our result carries the lesson to masking, and one step further. Masking is usually treated as unconditionally sound: proposals specify an order and a refresh strategy, rarely the build they were evaluated on. We find not only that the build decides whether the countermeasure works, but that on the build we study it decides the sign. At `-O3` the attack codeword masking was proposed to stop does not succeed in the first place, so the countermeasure defends against nothing — and the share it introduces is what makes the attack succeed. Establishing that an attack works on the build being shipped is therefore not a detail of the evaluation; it is a precondition for the countermeasure being worth deploying at all.

The practical reading is that codeword masking has to be justified against the specific attack, build and target it is meant to stop, rather than adopted across the board because an attack once worked somewhere. An attack demonstrated at `-O2` does not by itself argue for masking at `-O3`, where the venue it targeted may no longer be exploitable and where the mask may instead hand a replay adversary the diversity it lacks.

7 Conclusion

We claim no efficiency advantage over Dong et al. [20]: against $\mathbf{v} - \mathbf{u} \cdot \mathbf{y}$ their single trace requires far fewer measurements than our $R \approx 500$ replays. Our contribution is a different observation, that masking can be what enables the attack, and it holds only where the shares are recoverable. Recovering HQC-192 and HQC-256, whose five duplications leave a deeper accumulation ceiling than HQC-128's three (Section 5.3), will need a more efficient method still.

A pivotal open question raised by our findings is how to systematically evaluate a masking countermeasure, prior to deployment, to prevent unintended side-channel amplification. Our result is one instance of a general hazard: once the share is readable, refreshing it can help the attacker rather than the defender. Whether that hazard also arises outside a replay setting, and what a countermeasure has to avoid in order not to expose it, is what we think deserves study next.

Acknowledgments. We thank Guillaume Goy [28] for the discussions that led us to apply the post-decoding lift to the Reed–Muller encoder leakage, and Nathan Maillet [36] for answering our questions about their work and for providing code that let us reproduce their results. We also thank the anonymous reviewers, whose comments substantially improved the paper.

This work was partly supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (RS-2024-00399401, Development of Quantum-Safe Infrastructure Migration and Quantum Security Verification Technologies, 50%) and the Korea Planning & Evaluation Institute of Industrial Technology (KEIT) grant funded by the Korean government (MOTIE) (No. RS-2025-02634277, Development of Human-AI Teaming-based Autonomous Security System for Future Cyber Threats, 50%).

References

1. Alagic, G., Apon, D., Cooper, D., Dang, Q., Dang, T., Kelsey, J., Lichtinger, J., Liu, Y.K., Miller, C., et al.: Status report on the third round of the nist post-quantum cryptography standardization process (2022)
2. Alagic, G., Bros, M., Ciadoux, P., Cooper, D., Dang, Q., Dang, T., Kelsey, J., Lichtinger, J., Liu, Y.K., Miller, C., et al.: Status report on the fourth round of the nist post-quantum cryptography standardization process (2025)
3. Alissaei, B.: Permnet-rm: A constant-time reed-muller encoder for hqc via GF(2) zeta-transform butterfly decomposition (2026), available on request from b@vaultbytes.com
4. Aragon, N., Barreto, P., Bettaieb, S., Bidoux, L., Blazy, O., Deneuville, J.C., Gaborit, P., Ghosh, S., Gueron, S., Güneysu, T., et al.: Bike: bit flipping key encapsulation (2022)
5. Baïsse, C., Moran, A., Goy, G., Maillard, J., Aragon, N., Gaborit, P., Lecomte, M., Loiseau, A.: Secret and shared keys recovery on hamming quasi-cyclic with sasca. Designs, Codes and Cryptography **93**(6), 2137–2157 (2025)
6. Bernstein, D.J., Bhargavan, K., Bhasin, S., Chattopadhyay, A., Chia, T.K., Kannwischer, M.J., Kiefer, F., Paiva, T., Ravi, P., Tamvada, G.: Kyberslash: Exploiting secret-dependent division timings in kyber implementations. Cryptology ePrint Archive (2024)
7. Bernstein, D.J., Chou, T., Lange, T., von Maurich, I., Misoczki, R., Niederhagen, R., Persichetti, E., Peters, C., Schwabe, P., Sendrier, N., et al.: Classic mceliece: conservative code-based cryptography. NIST submissions **1**(1), 1–25 (2017)
8. Bernstein, D.J., Hülsing, A., Kölbl, S., Niederhagen, R., Rijneveld, J., Schwabe, P.: The sphincs+ signature framework. In: Proceedings of the 2019 ACM SIGSAC conference on computer and communications security. pp. 2129–2146 (2019)
9. Bernstein, D.J., Lange, T.: Post-quantum cryptography. Nature **549**(7671), 188–194 (2017)
10. Berthet, P.A.: Indepth analysis of a side-channel message recovery attack against frodokem. In: 2025 IEEE International Conference on Cyber Security and Resilience (CSR). pp. 666–671. IEEE (2025)
11. Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Seiler, G., Stehlé, D.: Crystals-kyber: a cca-secure module-lattice-based kem. In: 2018 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 353–367. IEEE (2018)
12. Bos, J.W., Gourjon, M.O., Renes, J., Schneider, T., Vredendaal, C.v.: Masking kyber: First-and higher-order implementations. IACR Transactions on Cryptographic Hardware and Embedded Systems **2021**(4), 173–214 (2021)
13. Chen, K.Y., Chen, J.P.: Masking floating-point number multiplication and addition of falcon: First-and higher-order implementations and evaluations. IACR Transactions on Cryptographic Hardware and Embedded Systems **2024**(2), 276–303 (2024)
14. Chen, L., Jordan, S., Liu, Y.K., Moody, D., Peralta, R., Perlner, R.A., Smith-Tone, D.: Report on post-quantum cryptography, vol. 12. US Department of Commerce, National Institute of Standards and Technology ... (2016)
15. Chen, M.S., Chien, T.Y., Chiu, C.M., Lin, H.H., Peng, C.T., Yang, B.Y.: Additive ffts for hqc on arm cortex-m4, revisited. Cryptology ePrint Archive (2026)
16. Colombier, B., Drăgoi, V.F., Cayrel, P.L., Grosso, V.: Profiled side-channel attack on cryptosystems based on the binary syndrome decoding problem. IEEE Transactions on Information Forensics and Security **17**, 3407–3420 (2022)

17. Dong, H., Guo, Q.: Multi-value plaintext-checking and full-decryption oracle-based attacks on hqc from offline templates. IACR Transactions on Cryptographic Hardware and Embedded Systems **2025**(4), 254–289 (2025)
18. Dong, H., Guo, Q.: Ot-pca: New key-recovery plaintext-checking oracle based side-channel attacks on hqc with offline templates. IACR Transactions on Cryptographic Hardware and Embedded Systems **2025**(1), 251–274 (2025)
19. Dong, H., Guo, Q.: Breaking optimized HQC: The first cache-timing full decryption oracle key-recovery attack in post-quantum cryptography. In: Advances in Cryptology – CRYPTO 2026. pp. 224–255. Springer (2026)
20. Dong, H., Guo, Q., Nabokov, D.: Single-trace key recovery attacks on HQC using valid and invalid ciphertexts. In: Advances in Cryptology – EUROCRYPT 2026. pp. 568–597. Springer (2026)
21. Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., Stehlé, D.: Crystals-dilithium: A lattice-based digital signature scheme. IACR Transactions on Cryptographic Hardware and Embedded Systems pp. 238–268 (2018)
22. Fouque, P.A., Hoffstein, J., Kirchner, P., Lyubashevsky, V., Pornin, T., Prest, T., Ricosset, T., Seiler, G., Whyte, W., Zhang, Z., et al.: Falcon: Fast-fourier lattice-based compact signatures over ntru. Submission to the NIST's post-quantum cryptography standardization process **36**(5), 1–75 (2018)
23. Fujisaki, E., Okamoto, T.: Secure integration of asymmetric and symmetric encryption schemes. In: Annual international cryptology conference. pp. 537–554. Springer (1999)
24. Gaborit, P., Aguilar-Melchor, C., Aragon, N., Bettaieb, S., Bidoux, L., Blazy, O., Deneuville, J.C., Persichetti, E., Zémor, G., Bos, J., et al.: Hamming quasi-cyclic (hqc). NIST Post-Quantum Standardization, 4th Round; National Institute of Standards and Technology: Gaithersburg, MD, USA (2025)
25. Goodwill, G., Jun, B., Jaffe, J., Rohatgi, P.: A testing methodology for side-channel resistance validation. In: NIST non-invasive attack testing workshop. vol. 7, pp. 115–136 (2011)
26. Goy, G., Loiseau, A., Gaborit, P.: Estimating the strength of horizontal correlation attacks in the hamming weight leakage model: a side-channel analysis on hqc kem. In: WCC 2022: The Twelfth International Workshop on Coding and Cryptography. p. WCC_2022_paper_48 (2022)
27. Goy, G., Loiseau, A., Gaborit, P.: A new key recovery side-channel attack on hqc with chosen ciphertext. In: International Conference on Post-Quantum Cryptography. pp. 353–371. Springer (2022)
28. Goy, G., Maillard, J., Gaborit, P., Loiseau, A.: Single trace hqc shared key recovery with sasca. IACR Transactions on Cryptographic Hardware and Embedded Systems **2024**(2), 64–87 (2024)
29. Haidar, C.A., Payet, Q., Tibouchi, M.: Crowhammer: full key recovery attack on falcon with a single rowhammer bit flip. In: Annual International Cryptology Conference. pp. 103–135. Springer (2025)
30. Hamburg, M., Hermelink, J., Primas, R., Samardjiska, S., Schamberger, T., Streit, S., Strieder, E., Van Vredendaal, C.: Chosen ciphertext k-trace attacks on masked cca2 secure kyber. IACR Transactions on Cryptographic Hardware and Embedded Systems pp. 88–113 (2021)
31. HQC Team: HQC: Official reference implementation (gitlab), commit `d622142`. `https://gitlab.com/pqc-hqc/hqc/-/commit/d622142a50f3ce6b6e1f5b15a5119d96c67194e0` (2026), accessed: 2026-01-14

32. Huang, Z., Wang, W., Zhou, X., Yu, Y.: Dpa-style attacks on hqc. IACR Transactions on Cryptographic Hardware and Embedded Systems **2026**(2), 928–952 (2026)
33. Karabulut, E., Aysu, A.: Falcon down: Breaking falcon post-quantum signature scheme through side-channel attacks. In: 2021 58th ACM/IEEE Design Automation Conference (DAC). pp. 691–696. IEEE (2021)
34. Lahr, N., Niederhagen, R., Petri, R., Samardjiska, S.: Side channel information set decoding using iterative chunking: Plaintext recovery from the "classic mceliece" hardware reference implementation. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 881–910. Springer (2020)
35. Lai, Z., Zhang, R., Zhang, Z., Hermelink, J., Schwarz, M., Pham, V.T., Parampalli, U.: You only decapsulate once: Ciphertext-independent single-trace passive side-channel attacks on hqc. Cryptology ePrint Archive (2025)
36. Maillet, N., Nugier, C., Migliore, V., Deneuville, J.C.: Key recovery from side-channel power analysis attacks on non-simd hqc decryption. In: Annual International Cryptology Conference. pp. 70–102. Springer (2025)
37. McCarthy, S., Howe, J., Smyth, N., Brannigan, S., O'Neill, M.: Bearz attack falcon: implementation attacks with countermeasures on the falcon signature scheme. Cryptology ePrint Archive (2019)
38. Melchor, C.A., Aragon, N., Bettaieb, S., Bidoux, L., Blazy, O., Deneuville, J.C., Gaborit, P., Persichetti, E., Zémor, G., Bourges, I.: Hamming quasi-cyclic (hqc). NIST PQC Round **2**(4), 13 (2018)
39. Merli, D., Stumpf, F., Sigl, G.: Protecting PUF error correction by codeword masking. Cryptology ePrint Archive, Report 2013/334 (2013), `https://eprint.iacr.org/2013/334`
40. Migliore, V., Gérard, B., Tibouchi, M., Fouque, P.A.: Masking dilithium: Efficient implementation and side-channel evaluation. In: International Conference on Applied Cryptography and Network Security. pp. 344–362. Springer (2019)
41. Miller, V.S.: Use of elliptic curves in cryptography. In: Conference on the theory and application of cryptographic techniques. pp. 417–426. Springer (1985)
42. Ngo, K., Dubrova, E., Guo, Q., Johansson, T.: A side-channel attack on a masked ind-cca secure saber kem implementation. IACR Transactions on Cryptographic Hardware and Embedded Systems pp. 676–707 (2021)
43. Paiva, T.B., Ravi, P., Jap, D., Bhasin, S., Das, S., Chattopadhyay, A.: Et tu, brute? side-channel assisted chosen ciphertext attacks using valid ciphertexts on hqc kem. In: International Conference on Post-Quantum Cryptography. pp. 294–321. Springer (2025)
44. Pessl, P., Primas, R.: More practical single-trace attacks on the number theoretic transform. In: International Conference on Cryptology and Information Security in Latin America. pp. 130–149. Springer (2019)
45. Primas, R., Pessl, P., Mangard, S.: Single-trace side-channel attacks on masked lattice-based encryption. In: International Conference on Cryptographic Hardware and Embedded Systems. pp. 513–533. Springer (2017)
46. Rao, A.R., Bhimasankaram, P.: Linear algebra. Springer (2000)
47. Ravi, P., Bhasin, S., Roy, S.S., Chattopadhyay, A.: On exploiting message leakage in (few) nist pqc candidates for practical message recovery attacks. IEEE Transactions on Information Forensics and Security **17**, 684–699 (2021)
48. Ravi, P., Chattopadhyay, A., D'Anvers, J.P., Baksi, A.: Side-channel and fault-injection attacks over lattice-based post-quantum schemes (kyber, dilithium): Sur-

vey and new results. ACM Transactions on Embedded Computing Systems **23**(2), 1–54 (2024)
49. Rivest, R.L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM **21**(2), 120–126 (1978)
50. Shor, P.W.: Algorithms for quantum computation: discrete logarithms and factoring. In: Proceedings 35th annual symposium on foundations of computer science. pp. 124–134. Ieee (1994)
51. Sim, B.Y., Kwon, J., Lee, J., Kim, I.J., Lee, T.H., Han, J., Yoon, H., Cho, J., Han, D.G.: Single-trace attacks on message encoding in lattice-based kems. IEEE Access **8**, 183175–183191 (2020)
52. Ueno, R., Xagawa, K., Tanaka, Y., Ito, A., Takahashi, J., Homma, N.: Curse of re-encryption: A generic power/em analysis on post-quantum kems. IACR Transactions on Cryptographic Hardware and Embedded Systems pp. 296–322 (2022)
53. Zhao, Y., Cheng, W., Qiao, Z., Liu, Y., Zhou, Y.: Rejection matters: Efficient non-profiling side-channel attack on ml-dsa via exploiting public templates. Cryptology ePrint Archive (2026)
54. Zhou, Y., Wang, W., Sun, Y., Yu, Y.: Rejected signatures' challenges pose new challenges: Key recovery of crystals-dilithium via side-channel attacks. IACR Transactions on Cryptographic Hardware and Embedded Systems **2025**(4), 817–847 (2025)

A CNN Bit-Classifier Architecture and Training

All CNN-based bit classifiers used in this paper (the per-bit message classifiers of Section 3 and the single- and multi-POI $\mathbf{v}-\mathbf{u}\cdot\mathbf{y}$ classifiers of Sections 4 and 5) share a single architecture and a single training recipe; only the input window and the number of profiling traces differ across venues. Each classifier is a binary 1-D CNN that maps a normalized trace window to a posterior over the value of one target bit $b \in \{0, 1\}$.

Table 6. 1-D CNN bit classifier. The three convolutional blocks use kernel sizes 7/5/3 and 32/64/128 channels; the adaptive average-pool makes the network agnostic to the input window length, so the same architecture serves both the 128-sample single-POI window and the concatenated 3×128-sample multi-POI window.

Stage	Layer	Output channels
Input	trace window ($1 \times L$)	1
Conv block 1	Conv1d(k=7) + BatchNorm + ReLU + MaxPool(2)	32
Conv block 2	Conv1d(k=5) + BatchNorm + ReLU + MaxPool(2)	64
Conv block 3	Conv1d(k=3) + BatchNorm + ReLU	128
Pooling	AdaptiveAvgPool1d(1) + Flatten	128
Head	Linear(128→64) + ReLU + Dropout(0.3)	64
Output	Linear(64→2) + softmax	2

Architecture. Table 6 summarizes the network. Each convolution is followed by batch normalization and ReLU, and the first two blocks downsample with stride-2 max-pooling. An adaptive average-pool collapses the temporal axis to length one, so the same architecture accepts both the 128-sample single-POI window ($W = 64$ half-width, $2W$ samples) and the concatenated multi-POI window ($K = 3$ windows, 3×128 samples) without change. A two-layer head with dropout 0.3 produces the two-class logits.

Training. Each window is z-score normalized per channel using the training-set mean and standard deviation. We train with AdamW (learning rate 10^{-3}) cosine-annealed over the full schedule, cross-entropy loss, and batch size 256; profiling traces are split 85/15 into training and internal validation, and the best validation epoch is kept. One classifier is trained per target bit and shared across all byte positions of its venue, giving 8 classifiers in total; for the message-recovery venue each is a 5-model ensemble trained with distinct random seeds and combined by soft-voting. The $\mathbf{v} - \mathbf{u} \cdot \mathbf{y}$ classifiers use 8 epochs and 20,000 profiling traces per bit; the message-recovery classifiers of Appendix B use 30 epochs over 3,000 training traces.

Inference. Each classifier outputs a per-trace softmax posterior. Under replay, the R posteriors of one fixed ciphertext are summed (soft-vote) before the $\arg\max$, the aggregation analyzed in Section 4.3 and Appendix C.

B Additional Details on Oracle-Substitution Experiments

We provide the experimental details for the oracle-substitution experiments discussed in Section 6. The key-recovery backends of OT-PCA [18] and MV-PC [17] are kept unchanged; only their side-channel oracle is replaced by the recovered-message oracle introduced in this work. Ciphertext construction, offline-template generation, reliability accumulation, and ISD post-processing follow the original experimental settings unless explicitly stated otherwise.

Beyond yielding the session key, single-trace message recovery *instantiates* the plaintext-checking oracle that the PCA family presupposes. A device that leaks m' on each decapsulation answers, for free, the PC query of OT-PCA [18] and the multi-value (MV-PC) query of its extension [17]. Those works otherwise serve both queries from precomputed offline templates and crafted decapsulations in order to recover the long-term key. We supply that oracle directly: from one (or a few replayed) traces, passively and on *valid* ciphertexts, we recover the full 128-bit message, reproduced end-to-end on STM32F415. The experiments below substitute that recovered message directly into the OT-PCA and MV-PC backends and confirm that the key recovery is preserved.

We do *not*, however, instantiate the full-decryption (FD) oracle of [17]: for HQC that is the *noisy* pre-RS-decoding codeword on the *decoder* side, carrying the key-dependent error pattern the FD attack needs, whereas our re-encryption venue exposes only the *clean* encoding of m', no richer than the message. This

PC/MV-PC-yes, FD-no boundary again follows the encoder-versus-decoder split of this work.

Recovered-Message Oracle. For each oracle query, the substituted oracle observes the RM-encoding venue during decapsulation and recovers the candidate 16-byte message m'. Concretely, the attack first estimates the 46-byte Reed–Solomon codeword from the 46 byte-level RM-encoding instances contained in the trace, and then applies the public RS decoder as a post-correction step. The resulting message estimate is used as the oracle response supplied to the original OT-PCA or MV-PC backend. The per-bit classifiers are profiled once, offline (Table 7), and that one profile then answers every oracle query the backend issues; the profiling cost is paid once, not per query. It is also not a large one: Table 1 shows RS-decoded recovery reaching 100% from 80 profiling traces, so the budget used here is comfortable rather than necessary.

Table 7. Training and validation setup for the recovered-message oracle.

Item	Value
Profiling traces (total)	5,000
Training split	3,000
Validation split	1,000
Test split	1,000
Classifier family	8 bit-level CNN ensembles per byte
Training epochs	30
Test byte accuracy	98.73%
Message success after RS post-correction	100.00%

Substitution into OT-PCA and MV-PC. For OT-PCA [18], the recovered message m' is converted into the binary plaintext-checking response expected by the original backend. For MV-PC [17], the recovered message is mapped to the corresponding multi-value oracle response used by the original MV-PC template lookup. No additional side-channel information is supplied to either backend.

The query budgets and numbers of validation keys follow the original experimental settings: 644 oracle queries per key over 120 keys for OT-PCA, and 276 oracle queries per key over 1,000 keys for MV-PC. Here, w_0 denotes the number of errors allowed in the initial post-processing collision search, following the notation of the original OT-PCA and MV-PC papers.

The substituted oracle closely tracks the original oracle instantiations: OT-PCA changes from 2.35 to 2.49 mean errors, and MV-PC changes from 3.74 to 3.81 mean errors. The remaining differences are expected because the validation keys, random seeds, target board, trigger alignment, and capture environment are not identical to those used in the original experiments.

Table 8. Oracle-substitution results compared with the original OT-PCA and MV-PC oracle experiments.

Attack	Oracle	Mean errors	$P_{\mathrm{succ}}(w_0 = 2)$	$P_{\mathrm{succ}}(w_0 = 3)$
OT-PCA ($t = 7$) [18]	Original	2.35	84.17%	92.50%
	This work	2.49	85.00%	90.00%
MV-PC ($t = 3$) [17]	Original	3.74	49.50%	63.00%
	This work	3.81	48.50%	61.30%

Overall, these results indicate that the recovered-message oracle can be used as a direct replacement for the side-channel oracle in the OT-PCA and MV-PC backends. The key-recovery behavior remains close to the original oracle-based experiments, while the oracle itself is instantiated from the RM-encoder message-recovery venue studied in this paper.

C Soft-Voting versus The Exact LLR Aggregation

The replay aggregation of Section 5.3 sums per-trace softmax posteriors (step (iii)), a *soft-vote* approximation of the Bayes-optimal log-odds rule of Eq. (7), which sums the per-trace log-likelihood ratios $L_r = \log[q(1 \mid t_r)/q(0 \mid t_r)]$. The two coincide only when the per-trace posteriors are well calibrated. Table 9 compares them on the multi-POI hybrid attack at $R = 500$: across all `cp` regions, masked and unmasked, the soft-vote and the exact LLR rule agree to within 0.2 percentage points, confirming that the CNN is calibrated enough for the softmax-sum reported in the main text to match the principled rule.

Table 9. Soft-vote (softmax-sum, SM) versus exact LLR-sum aggregation: per-cp average per-bit accuracy of the multi-POI hybrid attack at $R = 500$ (STM32F415, held-out test set). SM is the rule used in the main text (Section 5.3); LLR is $\sum_r L_r$ (Eq. (7)).

	Unmasked		Masked	
Region	SM	LLR	SM	LLR
`cp` = 0	1.0000	1.0000	1.0000	1.0000
`cp` = 1	0.9237	0.9231	1.0000	1.0000
`cp` = 2	0.8733	0.8746	1.0000	1.0000
all 384	0.9324	0.9326	1.0000	1.0000

D The Accumulation Ceiling at Five Duplications (HQC-192/256)

Section 4.4 attributes the residual per-bit error on $\mathbf{v} - \mathbf{u} \cdot \mathbf{y}$ to the duplication factor d: `expand_and_sum` writes `dst`$[k]$ += `stored`$[k + 128c]$ once per copy c, so the value exposed at copy c carries $c + 1$ secret bits at once. HQC-128 has $d = 3$; HQC-192 and HQC-256 have $d = 5$. If the explanation is right, the leak should weaken monotonically with c, and five copies should push it further than three.

Figure 10 tests this directly on HQC-192. We take the offset-0 bit of each copy, the five source bits that are summed into the *same* accumulator position `dst`[0], and plot the bit-conditioned Welch $|t|$ across the `expand_and_sum` execution window, one panel per copy, from 80,000 profiling traces on STM32F415 at 8 ns per sample.

Two features stand out. First, every panel peaks five times, at $\approx$ 3, 94, 219, 343 and 467 μs: those are the five writes of `dst`[0], and a bit keeps leaking at every write after its own because it stays in the accumulated sum. This is the same echo structure that Figure 4 shows for $d = 3$, and it is what the multi-POI strategy of Section 4.3 exploits.

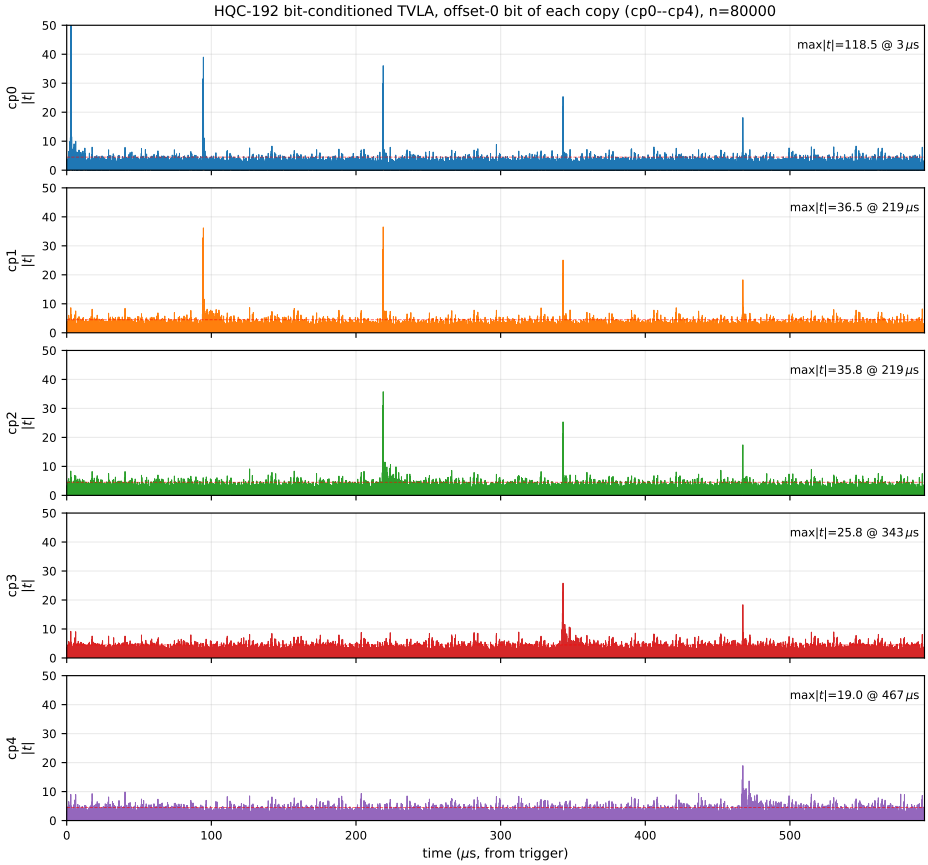


Fig. 10. Bit-conditioned Welch $|t|$ for the offset-0 bit of each of the five copies of an HQC-192 block, over the `expand_and_sum` window (STM32F415, 80,000 profiling traces, 8 ns/sample). The dashed line is the usual $|t| = 4.5$ threshold. The y axis is clipped at 50; where a panel exceeds it the peak is given in the annotation.

Second, and the point of the appendix, the peak $|t|$ falls monotonically with the copy index: 118.5, 36.5, 35.8, 25.8, 19.0 for `cp0` through `cp4`. The `cp0` write exposes a single bit and is by far the cleanest; by `cp4` the same accumulator position holds a five-bit sum and the leak is six times weaker. Raising d from 3 to 5 therefore deepens the ceiling rather than leaving it unchanged, which is what one should expect of the higher security levels.

E `gf_mul` Operand Leakage and Its Fragility

Both attacks of Dong et al. [20] consume an implementation contingency in the Galois-field multiplication `gf_mul`. To strip the codeword mask, VC-KRA-MDPC first recovers the per-decapsulation mask m' by attacking the systematic

Reed–Solomon encoder (Listing 1.1), where, in the words of [20], "`gf_mul` leaks significant information about its first operand," which "is directly derived from the secret message mask m', while the second operand is a known constant"; OT-FDA-CK's full-decryption oracle reads the inner Reed–Muller decoder outputs through the syndrome `gf_mul` the same way. The leakage is thus not structural but a consequence of *which operand carries the secret*, the dependence identified by Goy et al. [28], whose assessment we reproduce on F415 (Table 10): the first operand's 256-valued content is recoverable with a Fisher-LDA template (0.665), whereas the second operand and the output stay at the 1/256 chance level. Because `gf_mul` is commutative, a developer can swap the operands at no cost, and Goy et al. report that "after the swap ... we are not able to perform the [template] attack." We confirm this directly on the mask/message recovery underpinning [20]: in reference operand order, both per-byte and full 16-byte recovery reach 100%, whereas *swapping the operands* collapses recovery to 0.02 per byte and 0 overall (Table 11). A one-line, zero-overhead reordering thus removes the leakage on which Dong et al.'s codeword-mask bypass and full-decryption oracle both rely.

Table 10. `gf_mul` operand leakage on the *reference* HQC implementation (STM32F415, $N = 99\,993$ traces, Fisher-LDA templates, 90/10 split), reproducing Table 2 of Goy et al. [28]. Each entry is the probability of correctly predicting the indicated quantity from a single multiplication. Only the *first* operand exposes its 256-valued content; the second operand and output stay at chance ($1/256 \approx 0.004$ value, $1/9 \approx 0.111$ Hamming weight).

	Value (256-class)	Hamming weight (9-class)
Operand 0 (first)	**0.665**	0.976
Operand 1 (second)	0.014	0.456
Output	0.012	0.218

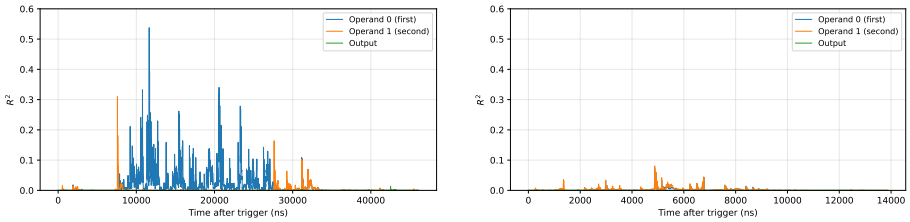
Table 11. Recovering the sensitive message byte from the Reed–Solomon encoder's `gf_mul` (STM32F415, target `msg[15]`), reproducing the codeword-mask recovery of [20, §6.1] in the reference operand order (*Normal*) and with the commutative operands *Swapped*. *Single-mult. accuracy* reads the 8-bit byte from one `gf_mul` (1/256 at chance); *per-byte* majority-votes over the ≈30 multiplications per byte; *full-message* requires all 16 bytes correct (over 250 held-out traces). Swapping leaves only 0.105 single-mult. accuracy, far below what majority voting needs.

	Normal	Swapped
Single-multiplication accuracy	1.000	0.105
Per-byte recovery (majority vote)	1.000	0.022
Full 16-byte message recovery	1.000	0.000

The same fragility is realized by the next HQC implementation. Chen et al. [15] rewrite both `gf_mul`-bearing venues for speed: a vectorized constant matrix–vector RS encoder $\mathbf{c} = \mathbf{u} \cdot [\, I_k \mid G \,]$ (replacing the per-byte `gf_mul` loop) and a packed, lazily reduced radix-16 decoder multiplier. Re-running the assessment on their multiplier (F415), the *value* leakage vanishes: all operands sit at the 1/256 chance level (Table 12) versus 0.665 on the reference, because the packed integer multiply no longer performs the shift/mask sequence that isolates an operand's bits. The value-based attacks of [20,28] are thereby starved. We do not overclaim immunity: the operands' Hamming weight still leaks (0.38–0.49), so a Goy-style SASCA is conceivable, but it would need a fresh factor graph for the new arithmetic; the existing value-based attacks do not transfer.

Table 12. The same `gf_mul` assessment as Table 10, now on the optimized radix-16 multiplier of Chen et al. [15] (STM32F415, $N = 99\,991$). The 256-valued content of *every* operand collapses to the 1/256 chance level (cf. 0.665 for the reference first operand), removing the value leakage that the `gf_mul`-based attacks consume; only Hamming-weight leakage persists.

	Value (256-class)	Hamming weight (9-class)
Operand 0 (first)	0.010	0.399
Operand 1 (second)	0.012	0.493
Output	0.012	0.383



(a) reference `gf_mul` (b) Chen et al. radix-16

Fig. 11. LRA bit-basis R^2 of the `gf_mul` operands across the multiplication window (STM32F415). On the reference implementation (a), operand 0 leaks strongly ($R^2_{\max} \approx 0.54$); on Chen et al.'s radix-16 multiplier (b) all operands collapse toward the noise floor ($R^2_{\max} \approx 0.08$), corroborating the value-recovery collapse of Tables 10 and 12.

F The PermNet-RM Constant-Time Encoder and Its Residual Leakage

The message-recovery venue of Section 3 exploits the `BITOMASK` idiom `mask = -((m » i) & 1)` that every reference HQC Reed–Muller encoder uses to select generator rows. PermNet-RM [3] eliminates this idiom at its root: it reformulates $\mathrm{RM}(1, m)$ encoding as the GF(2) zeta transform of a fixed indicator vector and computes it through a fixed-topology butterfly network of straight-line XOR-and-shift operations, in which the selector at every stage depends on the *register index*, not on the message. No message bit controls a branch or a select, so the data-dependent timing/`CMOV` channel of the `BITOMASK` loop is removed and the encoder is a constant-time, ABI-compatible drop-in for `reed_muller_encode()`. PermNet-RM proves 0-probing security after the first butterfly stage but explicitly notes a residual: under a bit-level probing model a bounded set of register positions still depends on a single message bit beyond that stage, to be handled by composing the encoder with Boolean masking.

PermNet-RM alone does not close the power leakage. We confirm this residual empirically. Figure 12 reports a fixed-versus-random TVLA on the PermNet-RM encoder on STM32F415 ($N = 1000$ traces per group, per message bit). The low-order bits are quiet ($\mathrm{Max}|t| \leq 9.4$ for bits 0–4), but the three high-order bits leak strongly ($\mathrm{Max}|t| = 22.0$, 174.1, 184.0 for bits 5, 6, 7), far above the ± 4.5 detection threshold. The constant-time reformulation removes the control-flow channel but not the Hamming-weight leakage of the message bits carried in the butterfly register state, exactly the limitation PermNet-RM anticipates. Deployed as-is, the encoder therefore still feeds the per-bit message-recovery pipeline of Section 3.

Output masking closes it. The residual is removed by masking the encoder as PermNet-RM recommends. Figure 13 repeats the TVLA on a masked PermNet-RM encoder under an output-masking ("outmask") scheme: in addition to Boolean masking of the input shares, a fresh 128-bit output mask R is XORed into the codeword store throughout the observed encode, so every value written during the triggered region carries $\mathrm{encode}(m) \oplus R$, a one-time-pad-blinded, message-independent quantity, and R is stripped only after the observation window, before the codeword is returned. Every message bit now falls below the detection threshold ($\mathrm{Max}|t| \leq 4.1$ across all eight bits), with no residual peak on bits 5–7. Composing PermNet-RM's constant-time structure with output masking thus closes the RM-encoder leakage point that our message-recovery attack targets.

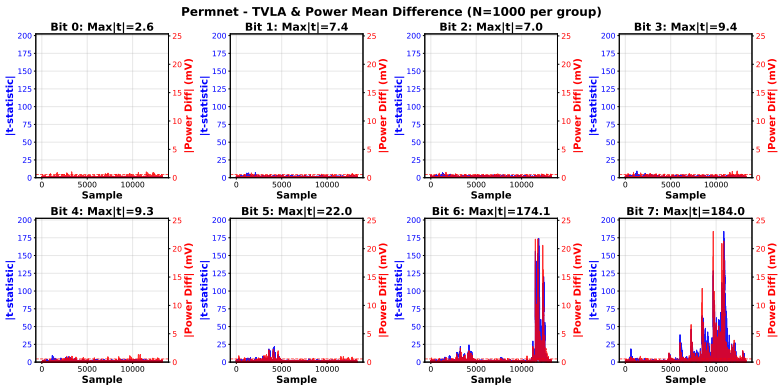


Fig. 12. TVLA of the (unmasked) PermNet-RM encoder on STM32F415 (1,000 traces per bit value, other bits random). Blue/left axis: Welch t statistic; red/right axis: mean power difference. Bits 0–4 stay near the noise floor, but bits 5, 6, 7 leak strongly ($\mathrm{Max}|t| = 22.0, 174.1, 184.0$): the constant-time reformulation removes the control-flow channel but not the Hamming-weight leakage of the message bits in the butterfly register state.

G Key Recovery from $\mathbf{v} - \mathbf{u} \cdot \mathbf{y}$

For completeness we recall the key-recovery step of Maillet et al. [36], which turns a recovered $\mathbf{v} - \mathbf{u} \cdot \mathbf{y}$ into the long-term secret. Work in the ambient ring $\mathcal{R} = \mathbb{F}_2[X]/(X^n - 1)$, and write the recovered value as $\mathbf{V} = \mathbf{v} - \mathbf{u} \cdot \mathbf{y}$. Since the ciphertext $(\mathbf{u}, \mathbf{v})$ is public, the attacker isolates the secret-dependent product exactly,

$$\mathbf{u} \cdot \mathbf{y} \;=\; \mathbf{v} - \mathbf{V}. \tag{9}$$

Invertible $\mathbf{u}$ *(the typical case).* The polynomial multiplication is a circulant matrix–vector product, $\mathbf{u} \cdot \mathbf{y} = \mathbf{y} \times \mathrm{rot}(\mathbf{u})^\top$, where $\mathrm{rot}(\mathbf{u}) \in \mathbb{F}_2^{n \times n}$ is the circulant matrix of $\mathbf{u}$. When $\mathbf{u}$ is invertible in $\mathcal{R}$ we have $\gcd(\mathbf{u}, X^n - 1) = 1$, so the extended Euclidean algorithm returns a, b with $a\,\mathbf{u} + b\,(X^n - 1) = 1$; the attacker takes $\mathbf{u}^{-1} = a \bmod (X^n - 1)$ and recovers

$$\mathbf{y} \;=\; \mathbf{u}^{-1} \cdot \big(\mathbf{v} - \mathbf{V}\big) \;=\; \mathbf{u}^{-1} \cdot (\mathbf{u} \cdot \mathbf{y}). \tag{10}$$

The remaining secret follows from the public key $\mathbf{s} = \mathbf{x} + \mathbf{h} \cdot \mathbf{y}$ as $\mathbf{x} = \mathbf{s} - \mathbf{h} \cdot \mathbf{y}$, yielding the full key $\mathrm{sk} = (\mathbf{x}, \mathbf{y})$. A single ring inversion thus suffices.

Non-invertible $\mathbf{u}$ *(the rank-deficient case).* When $\mathbf{u}$ is not invertible, $\mathrm{rot}(\mathbf{u})$ has rank $r < n$ and $\mathbf{y}$ is no longer uniquely determined by Eq. (9); the attacker can nonetheless recover r of its bits. By the rank decomposition [46, Th. 3.6.5] there exist a full-column-rank $\mathbf{P} \in \mathbb{F}_2^{n \times r}$, a permutation matrix $\mathbf{Q}$, and $\mathbf{C} \in \mathbb{F}_2^{r \times (n-r)}$ with

$$\mathrm{rot}(\mathbf{u}) \;=\; \mathbf{P} \times [\,\mathbf{I}_r : \mathbf{C}\,] \times \mathbf{Q}. \tag{11}$$

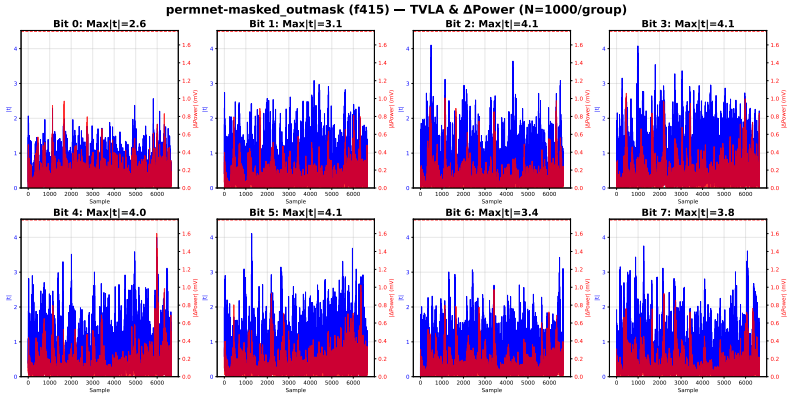


Fig. 13. TVLA of the masked PermNet-RM encoder (Boolean input-share masking + a fresh 128-bit output mask R) on STM32F415, same setting as Figure 12. Note the rescaled $|t|$ axis: every bit now stays below the ± 4.5 threshold ($\mathrm{Max}|t| \le 4.1$), the strong bit-5/6/7 leakage of the unmasked encoder having vanished.

Writing $\mathbf{P_L^{-1}}$ for a left inverse of $\mathbf{P}$ and right-multiplying Eq. (9) by $\left(\mathbf{P_L^{-1}}\right)^\top$ telescopes the system,

$$\begin{aligned}\left(\mathbf{v} - (\mathbf{v} - \mathbf{u}\cdot\mathbf{y})\right) \times \left(\mathbf{P_L^{-1}}\right)^\top &= \left(\mathbf{y} \times \mathrm{rot}(\mathbf{u})^\top\right) \times \left(\mathbf{P_L^{-1}}\right)^\top \\ &= \left(\mathbf{y} \times \mathbf{Q}^\top \times [\,\mathbf{I}_r : \mathbf{C}\,]^\top \times \mathbf{P}^\top\right) \times \left(\mathbf{P_L^{-1}}\right)^\top \\ &= \mathbf{y} \times \mathbf{Q}^\top \times [\,\mathbf{I}_r : \mathbf{C}\,]^\top \;=\; \mathbf{y}' \times [\,\mathbf{I}_r : \mathbf{C}\,]^\top,\end{aligned} \tag{12}$$

where $\mathbf{y}' = \mathbf{y} \times \mathbf{Q}^\top$. This exposes the r leading bits of $\mathbf{y}'$, the same bits as $\mathbf{y}$ but permuted, whose order is restored by left-multiplying with $\mathbf{Q}^{-1} = \mathbf{Q}^\top$; the remaining $n - r$ bits are brute-forced.

The brute-force cost is at most one bit. The feasibility of that brute force is governed by the rank of $\mathrm{rot}(\mathbf{u})$, given by [36, Lemma 1]: with $\mathbb{F} = 1 + X + \cdots + X^{n-1}$,

$$\mathrm{rk}\big(\mathrm{rot}(\mathbf{u})\big) = \begin{cases} 0 & \text{if } \mathbf{u} = 0, \\ 1 & \text{if } \mathbf{u} = \mathbb{F}, \\ n-1 & \text{if } \mathbf{u} \neq 0 \text{ and } \omega(\mathbf{u}) \text{ even}, \\ n & \text{otherwise } (\mathbf{u} \neq \mathbb{F},\ \omega(\mathbf{u}) \text{ odd}). \end{cases} \tag{13}$$

The degenerate cases $\mathbf{u} \in \{0, \mathbb{F}\}$ essentially never arise for a freshly drawn $\mathbf{u}$, so $r \in \{n-1, n\}$: either $\mathbf{u}$ is invertible ($r = n$) or exactly $n - r = 1$ bit must be brute-forced. The empirical split reported in [36] matches this: of 4,600,000 sampled $\mathbf{u}$, 2,300,148 were invertible and the remainder leaked $n - 1$ bits.

In every case a *bit-exact* $\mathbf{v} - \mathbf{u}\cdot\mathbf{y}$, which our masked-mode attack attains (Section 5), yields the long-term key $(\mathbf{x}, \mathbf{y})$ from a single ring inversion, up to at most one brute-forced bit.