

Breaking First-Order Masked Kyber with Belief Propagation Alone

Vahid Jahandideh, Lejla Batina

Radboud University, Nijmegen, The Netherlands

{v.jahandideh,lejla}@cs.ru.nl

Abstract. We recover the long-term key of a first-order masked KYBER768 implementation on an ARM Cortex-M4 from about 2200 decapsulation power traces, using only authentic ciphertexts that pass ciphertext-validity checks. The secret is freshly shared before every decapsulation, and the attack combines weak second-order leakage from both shares. Under the evaluated leakage model, even exact Hamming weights of both shares provide about 97% less mutual information about a decryption-noise coefficient than the exact Hamming weight of the unmasked coefficient.

A profiled neural network estimates Hamming-weight distributions for the two arithmetic shares of each noisy message coefficient. We combine these distributions into likelihoods on the decryption noise, which is a known affine function of the secret and the public-key error. A belief-propagation (BP) solver accumulates these likelihoods across traces, preserving their non-Gaussian shapes while approximating the aggregate interference from other key coordinates by a Gaussian. An Onsager correction from approximate message passing corrects for feedback between each factor and the current beliefs. The solver recovers all 1536 secret and error coefficients without lattice reduction.

At the evaluated trace budgets, least squares performs no better than the all-zero estimate, and a greedy inequality solver fails even with exact Hamming weights. The evaluated primal-lattice cascade does not reduce the trace requirement: below BP’s recovery threshold, its output remains too diffuse to enable practical lattice recovery. At 2000 traces, BP-derived hints lower the estimated BKZ block size from 624 to 505, leaving the lattice attack impractical.

Keywords: ML-KEM · Side-channel analysis · Belief propagation · Lattice attacks

1 Introduction

The Module-Lattice-Based Key-Encapsulation Mechanism (ML-KEM), standardized in FIPS 203 [Nat24], is derived from CRYSTALS-KYBER [BDK⁺21]. Its black-box security rests on the hardness of Module Learning With Errors (MLWE), but this guarantee does not protect implementations that leak through physical side channels. Leakage may reveal secret-dependent values directly, or provide partial information that makes recovering the long-term key easier than solving the public MLWE instance alone [DDGR20, MN23, DGHK23, HSMP25]. The standard countermeasure is masking: every sensitive intermediate is split into random shares, so that no single share depends on the secret. First-order masked implementations of lattice-based KEMs exist for embedded targets [OSPG18, HKL⁺22]. Against them, an attacker must combine the leakage of two shares from the same execution, and this second-order leakage is weak.

This paper asks whether such weak leakage can still be accumulated into full key recovery. We answer affirmatively for the first-order masked KYBER768 implementation of [HKL⁺22] on an ARM Cortex-M4. Using only valid ciphertexts, we recover all 1536

coefficients of the secret and the public-key error from about 2200 decapsulation traces. The key is to keep the leakage probabilistic throughout. A profiled neural network reads the Hamming weights of the two shares of each decryption-noise coefficient, which we combine into a likelihood on that coefficient, and a belief-propagation solver fuses the likelihoods of all traces. The solver keeps each likelihood intact and approximates only the interference from the other key coordinates as Gaussian [DMM09, Ran11].

Solvers that discard the shape of each likelihood fail on this leakage. Least squares keeps only its mean and collapses onto the all-zero key. The greedy solver of [RPJ⁺24], designed for decryption-noise leakage from authentic ciphertexts, keeps only an interval per observation. It fails at our trace budget even with exact Hamming weights, and its authors' own masked simulation needs about 6×10^4 ciphertexts. We also evaluate whether a primal lattice attack can complete a partial recovery. It cannot in our setting: below the recovery threshold, the solver's output is too diffuse, and the lattice step becomes cheap only when the solver is about to recover the key on its own.

1.1 Threat Model

We consider a standard *profiled* attacker [CRR02] whose goal is the long-term secret $\mathbf{s}$. In a profiling phase, the attacker records masked decapsulations with known share values and trains a neural network that maps each trace segment to distributions of the two share Hamming weights. We profile on the target board itself, using decapsulations separate from the attack traces (Section 3.2). In the attack phase, the attacker encapsulates to the public key with the standard procedure, keeps each message and its encryption randomness, submits the ciphertext for decapsulation, and records one power trace per query. As in [RPJ⁺24], the attack thus uses only valid ciphertexts; we call them *authentic*, in contrast to ciphertexts malformed to amplify leakage [HHP⁺21]. It therefore induces no decapsulation failures, which could trigger key-refresh or reset countermeasures [XPRO20].

Each query t gives a noisy view of the *decryption noise* $\mathbf{w}_t$, whose $n = 256$ coefficients satisfy the integer affine relation (Section 2.2)

$$\mathbf{w}_t = \mathbf{M}_t [\mathbf{s}, \mathbf{e}] + \mathbf{z}_t, \quad (1)$$

where $\mathbf{e}$ is the public-key error and $\mathbf{M}_t, \mathbf{z}_t$ are known from the ciphertext and its encryption randomness. For KYBER768, $[\mathbf{s}, \mathbf{e}]$ has $2kn = 1536$ short coefficients, and every query adds 256 relations in these same unknowns. Unlike the approximate hints of [DDGR20], the observations are not noisy linear functions of the key: the device leaks only the Hamming weights of two masked shares of the value that carries $\mathbf{w}_t$, as described next, and the attack must first turn them into a likelihood on each decryption-noise coefficient.

1.2 What Masking Changes

Our target is the open-source first-order masked KYBER768 implementation of [HKL⁺22] in its optimized STM32F4 build. We add a refresh of the secret's arithmetic sharing before every decapsulation, preserving the long-term key while preventing repeated queries of the same ciphertext from exposing fixed share values at the targeted loads.

Suppressing the query index t , the noisy message is represented by two arithmetic shares and is never formed in the clear:

$$\mathbf{m}' = \lceil \mathbf{m} \rceil_{2 \rightarrow q} + \mathbf{w} = \mathbf{m}'_1 + \mathbf{m}'_2 \pmod{q}. \quad (2)$$

Here $q = 3329$, and the known message encoding maps each bit to 0 or 1665. Under the uniform-share model, each share coefficient is uniform and independent of the corresponding decryption-noise coefficient, so its Hamming weight alone reveals nothing about it. Our

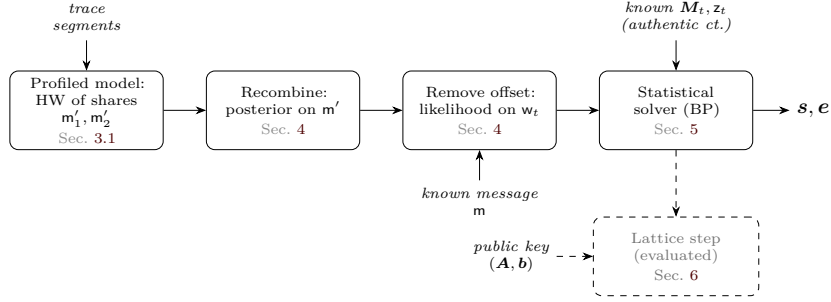


Figure 1: Attack workflow. The dashed branch denotes the evaluated lattice cascade.

attack combines information from both shares within one execution and is second-order under this model.

A two-head neural network estimates a Hamming-weight distribution for each share load during message decoding (Figure 2). We allocate class probabilities uniformly among compatible share values, convolve the resulting distributions modulo q , and remove the known message offset to obtain an approximate decryption-noise likelihood (Section 4).

Under the evaluated leakage model, exact Hamming weights of both shares provide only 0.079 bits of mutual information about a decryption-noise coefficient, compared with 2.77 bits for the Hamming weight of the unmasked coefficient: approximately 97% less information before measurement noise (Section 4.1). These weak observations nevertheless accumulate across queries because they constrain the same secret and public-key error through Eq. (1). Figure 1 summarizes the attack.

1.3 Soft Hints and the Lattice Step

Earlier side-channel cascades use a statistical solver as the front end of a lattice attack [HMS⁺23]. Information on both $\mathbf{s}$ and $\mathbf{e}$ shortens the planted vector of the primal embedding of $\mathbf{b} = \mathbf{A}\mathbf{s} + \mathbf{e} \bmod q$ [DDGR20], whereas it does not similarly help the dual attack [CMST25, PS24]. We therefore feed BP’s marginals to the primal attack as approximate hints, plus perfect hints for the few coordinates that BP is certain of (Section 6). This does not reduce the number of traces: below the recovery threshold, the marginals are calibrated but too diffuse, and the lattice step becomes cheap only when BP is about to recover the key on its own. This is a negative result for the evaluated instance, not a claim that lattice assistance can never help against masked implementations.

1.4 Related Work

Chosen-ciphertext attacks on masked Kyber. A prominent line of attacks on KYBER crafts the ciphertexts such that an intermediate of the decapsulation depends on only a few key coefficients, and the attacker recovers that intermediate from the leakage, often with profiled deep learning [RRCB19, HHP⁺21, NDGJ21, BNGD23, DNGW23, WBD24]. Closest to our target, Wang et al. [WBD24] recover the key of the bitsliced first- to third-order masked KYBER768 of [BC22] on an ARM Cortex-M4, from leakage of its Boolean-to-arithmetic conversion. Such ciphertexts are not valid encryptions: they fail the re-encryption check of the Fujisaki–Okamoto transform, so detection countermeasures that reject malformed ciphertexts block them [RPJ⁺24].

Inequalities and hints. Several attacks recover lattice-KEM secrets from decryption failures, injected faults, or side-channel leakage by converting them into inequalities and solving them statistically [PP21, HPP21, Del22, QWC⁺24, RPJ⁺24], and Hermelink et

al. [HMS⁺23] combine such belief propagation with lattice reduction. A complementary line studies how exact, approximate, or independent hints change the hardness of LWE and MLWE [DDGR20, DGHK23, MN23, LFP25]; we use it to evaluate the lattice step (Section 6). Hermelink et al. [HSMP25] generalize most of these hints and inequalities to *distribution hints*, arbitrary distributions on a linear function of the key, which they solve by belief propagation or by a greedy approximation of it. In simulation, they apply them to the Hamming weight of the *unmasked* decryption noise and need half as many ciphertexts as the inequalities of [RPJ⁺24]. Our observations are distribution hints of this kind, but they come from masked leakage, so they are much weaker and far more numerous: complete recovery uses about 5.6×10^5 of them, each involving all 1536 key coordinates. Their belief propagation computes every message exactly and stores one per edge, and they note that it is limited to moderate numbers of hints [HSMP25].

Message passing. Belief propagation drives soft analytical side-channel attacks [VGS14], including single-trace attacks on masked lattice-based decryption [PPM17] and the few-trace attacks of [HHP⁺21]. Their factor graphs are sparse: they follow the operations of the implementation. Our graph is dense, since every factor involves all 1536 key coordinates. We therefore follow relaxed belief propagation and approximate message passing for dense linear mixing [Kab03, DMM09, Ran10, Ran11], which treat the interference of the other coordinates as Gaussian, while keeping each measured likelihood non-Gaussian and each key coordinate discrete (Section 5.2).

Other targets and implementations. Other attacks on KYBER target the NTT-domain secret multiplication [BBB⁺24, KT24, WXT25], a masked hardware implementation [JD25], or arithmetic-to-Boolean conversions [BDV21]. Masked implementations range from the table-based first-order design of [HKL⁺22], which we target, to first- and higher-order [BGR⁺21] and bitsliced [BC22] designs. Our evaluation relies on leakage from individual arithmetic coefficient shares; its transfer to other data layouts and conversion routines remains untested.

1.5 Contributions

We make the following contributions.

- **Complete recovery from a first-order masked implementation.** We demonstrate a profiled power side-channel attack on the optimized first-order masked KYBER768 implementation of [HKL⁺22] on an STM32F4, with the secret sharing refreshed before every decapsulation. Using only authentic ciphertexts, we recover the secret s and public-key error e , all 1536 coefficients, from about 2200 traces. To our knowledge, this is the first complete recovery from this optimized implementation on real hardware under these conditions.¹
- **Scalable BP for masked decryption-noise likelihoods.** Building on distribution-hint BP [HSMP25], we combine likelihoods reconstructed from share leakage through the affine dependence of the decryption noise on the secret and public-key error. The solver retains the non-Gaussian likelihood shapes while approximating aggregate interference as Gaussian. Onsager corrections from approximate message passing approximately account for each factor’s own feedback without storing a message per edge [DMM09, Ran11]. In our evaluation, these corrections reduce the trace requirement for complete recovery from about 3500 to about 2200 (Section 5.3).

¹Our attack code will be made public.

- **An analysis of the least-squares and inequality baselines.** At the evaluated trace budgets, least squares performs no better than the all-zero estimate, while the greedy inequality solver of [RPJ⁺24] fails even with exact share Hamming weights. Least squares averages away weak likelihood variations. The greedy solver uses weakly informative intervals, with finite profiling introducing bounds that can exclude the true decryption noise. These comparisons show the benefit of retaining the likelihood shape in our attack.
- **An evaluation of lattice assistance.** We estimate whether a primal lattice attack can complete partial BP recovery. For the evaluated instance and hint framework, the estimated cost becomes potentially feasible only when BP is already close to full recovery, providing no reduction in the required trace count (Section 6).

2 Preliminaries

Polynomial ring. Throughout, $n = 256$ and $q = 3329$, and $\mathbb{Z}_q$ denotes the integers modulo q . For integers x , $x \bmod^+ a$ denotes the representative of x modulo a in $[0, a - 1]$; for odd a , $x \bmod^\pm a$ denotes the representative in $[-(a - 1)/2, (a - 1)/2]$. KYBER works in the ring $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$. We represent each element by a polynomial of degree below n , writing polynomials in lowercase sans-serif and their coefficients in *italic*: $\mathbf{f} = \sum_{i=0}^{n-1} f_i X^i$. We identify $\mathbf{f}$ with its coefficient column vector $(f_0, \dots, f_{n-1})^\top$. Addition is coefficient-wise. The $\mathbf{h} = \mathbf{f}\mathbf{g}$ product is given by the negacyclic convolution

$$h_i = \sum_{j=0}^{n-1} g_j f_{(i-j) \bmod^+ n} \text{sign}(i - j), \quad 0 \leq i < n,$$

where $\text{sign}(i - j) = 1$ if $i \geq j$ and -1 otherwise, and coefficient arithmetic is in $\mathbb{Z}_q$. The sign accounts for the relation $X^n = -1$. For fixed $\mathbf{f}$, define the negacyclic multiplication matrix $\text{Neg}(\mathbf{f}) \in \mathbb{Z}_q^{n \times n}$ by

$$\text{Neg}(\mathbf{f})_{i,j} = f_{(i-j) \bmod^+ n} \text{sign}(i - j), \quad 0 \leq i, j < n.$$

Thus, the coefficient vector of $\mathbf{f}\mathbf{g}$ is $\text{Neg}(\mathbf{f})\mathbf{g}$. Although KYBER implements multiplication through the number-theoretic transform, the equivalent negacyclic representation is more convenient for our analysis.

Vectors. Vectors of polynomials are written in bold lowercase and matrices in bold uppercase. For the module rank k ($k = 3$ for KYBER768; Table 1), a vector $\mathbf{u} = (u_0, \dots, u_{k-1}) \in \mathcal{R}_q^k$ is identified with the concatenation of the coefficient vectors of its k components, a vector of length kn . For $\mathbf{u}, \mathbf{v} \in \mathcal{R}_q^k$, we write $\langle \mathbf{u}, \mathbf{v} \rangle = \sum_{j=0}^{k-1} u_j v_j \in \mathcal{R}_q$.

Probability. For a discrete random variable Y , we write p_Y for its probability mass function (PMF), so that $p_Y(y) = \Pr[Y = y]$, and, for a conditioning variable Z , $p_Y(y | z) = \Pr[Y = y | Z = z]$. A hat denotes an estimate throughout: $\hat{y}$ is an estimate of y , and $\hat{p}_Y$ is a PMF produced by a model, such as a classifier, in place of the true p_Y . We write $y \stackrel{\$}{\leftarrow} \chi$ when y is sampled from a distribution χ , and $\mathbf{f} \stackrel{\$}{\leftarrow} \chi(\mathcal{R}_q)$ when the coefficients of $\mathbf{f}$ are sampled independently from χ ; $\chi(\mathcal{R}_q^k)$ and $\chi(\mathcal{R}_q^{k \times k})$ extend this coefficient-wise to vectors and matrices of polynomials. $\mathcal{U}_a$ is the uniform distribution on $\{0, \dots, a - 1\}$. The centered binomial distribution χ_η with parameter η is the distribution of the difference of two independent $\text{Bin}(\eta, 1/2)$ random variables; for $Y \sim \chi_\eta$,

$$p_Y(y) = \binom{2\eta}{\eta + y} 2^{-2\eta}, \quad y \in \{-\eta, \dots, \eta\}.$$

Modular convolution. If Y_1 and Y_2 are independent random variables on $\mathbb{Z}_q$, the PMF of their sum in $\mathbb{Z}_q$ is the cyclic convolution

$$(p_{Y_1} *_q p_{Y_2})(j) = \sum_{i=0}^{q-1} p_{Y_1}(i) p_{Y_2}((j-i) \bmod^+ q), \quad 0 \leq j < q.$$

We evaluate it with the length- q complex discrete Fourier transform $\mathcal{F}$, using $\mathcal{F}(p_{Y_1} *_q p_{Y_2}) = \mathcal{F}(p_{Y_1}) \cdot \mathcal{F}(p_{Y_2})$ with an element-wise product. Section 4 uses this operation to combine the two share-value PMFs.

Information measures. We write $H(Y)$ for the Shannon entropy of Y and $I(Y; Z)$ for the mutual information between Y and Z , both in bits. When the conditional PMF $p_Y(\cdot | z)$ is available only through a model $\hat{p}_Y(\cdot | z)$, we use the perceived information [BHM⁺19]

$$\text{PI}(Y; Z) = H(Y) - \mathbb{E}[-\log_2 \hat{p}_Y(Y | Z)],$$

where the expectation is over the true joint distribution of (Y, Z) and is estimated on held-out data. The subtracted term is the cross-entropy of the model, which is at least $H(Y | Z)$; hence $\text{PI}(Y; Z) \leq I(Y; Z)$, with equality when $\hat{p}_Y = p_Y$.

Compression and decompression. For moduli $a, b \geq 2$ and an integer x , we write

$$\lceil x \rceil_{a \rightarrow b} = \left\lceil \frac{b}{a} (x \bmod^+ a) \right\rceil \bmod^+ b,$$

where $\lceil \cdot \rceil$ rounds to the nearest integer, with ties rounded up. The map compresses when $a > b$ and decompresses when $a < b$, and is applied coefficient-wise to polynomials and vectors. In particular, $\lceil \cdot \rceil_{2 \rightarrow q}$ encodes a message bit as $0 \mapsto 0$ and $1 \mapsto \lceil q/2 \rceil = 1665$, and $\lceil \cdot \rceil_{q \rightarrow 2}$ decodes it.

2.1 Kyber Specification

The public-key encryption (PKE) component of KYBER consists of **Key-Gen**, **Encrypt**, and **Decrypt** (Algorithm 1) [Nat24, BDK⁺21], with the parameters of Table 1. Messages are 256-bit strings, identified with polynomials in $\mathcal{R}_2 = \mathbb{Z}_2[X]/(X^n + 1)$.

Algorithm 1 Kyber public-key encryption (IND-CPA)

```

1: function Key-Gen( $k$ )
2:    $A \xleftarrow{\$} \mathcal{U}_q(\mathcal{R}_q^{k \times k})$ 
3:    $s \xleftarrow{\$} \chi_{\eta_1}(\mathcal{R}_q^k), \quad e \xleftarrow{\$} \chi_{\eta_1}(\mathcal{R}_q^k)$ 
4:    $b \leftarrow As + e$ 
5:   return  $pk := (A, b), sk := s$ 

6: function Encrypt( $pk, m, \text{seed}$ )
7:    $(r, e_1, e_2) \xleftarrow{\text{seed}} (\chi_{\eta_1}(\mathcal{R}_q^k), \chi_{\eta_2}(\mathcal{R}_q^k), \chi_{\eta_2}(\mathcal{R}_q))$  ▷ seed drives a deterministic PRNG
8:    $u \leftarrow A^T r + e_1$ 
9:    $v \leftarrow \langle b, r \rangle + e_2 + \lceil m \rceil_{2 \rightarrow q}$  ▷  $0 \mapsto 0, 1 \mapsto \lceil q/2 \rceil$ 
10:   $c_1 \leftarrow \lceil u \rceil_{q \rightarrow D_u}, \quad c_2 \leftarrow \lceil v \rceil_{q \rightarrow D_v}$ 
11:  return  $c := (c_1, c_2)$ 

12: function Decrypt( $c_1, c_2, sk$ )
13:   $u' \leftarrow \lceil c_1 \rceil_{D_u \rightarrow q}, \quad v' \leftarrow \lceil c_2 \rceil_{D_v \rightarrow q}$ 
14:   $m' \leftarrow v' - \langle u', s \rangle$ 
15:   $m \leftarrow \lceil m' \rceil_{q \rightarrow 2}$ 
16:  return  $m$  ▷ 256-bit message

```

Table 1: ML-KEM parameter sets.

Parameter set	k	η_1	η_2	D_u	D_v
ML-KEM-512	2	3	2	2^{10}	2^4
ML-KEM-768	3	2	2	2^{10}	2^4
ML-KEM-1024	4	2	2	2^{11}	2^5

The PKE scheme is only IND-CPA secure. ML-KEM (Algorithm 2) applies the Fujisaki–Okamoto transform with implicit rejection to obtain IND-CCA2 security, using hash functions $\mathbf{G}$, $\mathbf{H}$ and $\mathbf{J}$ (SHA3-512, SHA3-256 and SHAKE256). Two properties are relevant here. First, **Encaps** derives the encryption randomness deterministically from the message, $(K, \text{seed}) = \mathbf{G}(\mathbf{m} \parallel \mathbf{H}(pk))$. An attacker who runs **Encaps** and retains $\mathbf{m}$ therefore knows $\mathbf{r}, \mathbf{e}_1, \mathbf{e}_2$ and can recompute the decompression errors defined in Section 2.2. Second, **Decaps** re-encrypts the recovered message and compares the result with the received ciphertext, so any ciphertext that is not the honest encryption of the message it decrypts to, such as a modified one, fails the check. **Decaps** then performs implicit rejection: instead of an error, it returns the pseudorandom key $\mathbf{J}(z \parallel c)$, where z is a secret random value stored with the decapsulation key. Our attack submits only authentic ciphertexts (Section 1.1), so, except with negligible decryption-failure probability, it never takes this branch.

Algorithm 2 ML-KEM encapsulation and decapsulation (simplified)

```

1: function Encaps( $pk$ )
2:    $\mathbf{m} \xleftarrow{\$} \mathcal{U}_2(\mathcal{R}_2)$ 
3:    $(K, \text{seed}) \leftarrow \mathbf{G}(\mathbf{m} \parallel \mathbf{H}(pk))$ 
4:    $c \leftarrow \text{Encrypt}(pk, \mathbf{m}, \text{seed})$ 
5:   return  $(K, c)$ 

6: function Decaps( $c, sk, pk, z$ )
7:    $(c_1, c_2) \leftarrow c$ 
8:    $\mathbf{m}^* \leftarrow \text{Decrypt}(c_1, c_2, sk)$  ▷ leak point (Algorithm 3)
9:    $(K^*, \text{seed}^*) \leftarrow \mathbf{G}(\mathbf{m}^* \parallel \mathbf{H}(pk))$ 
10:   $c^* \leftarrow \text{Encrypt}(pk, \mathbf{m}^*, \text{seed}^*)$  ▷ re-encryption
11:  if  $c^* \neq c$  then
12:     $K^* \leftarrow \mathbf{J}(z \parallel c)$  ▷ implicit rejection
13:  return  $K^*$ 

```

Our target implementation [HKL⁺22] follows round-3 KYBER, whose PKE and parameters coincide with ML-KEM; the two differ only in the hashing around the transform (round-3 KYBER hashes $\mathbf{m}$ in **Encaps** and derives the session key with an additional KDF), which does not affect the attack.

2.2 The Target: Decryption Noise as a Linear System

The secret $\mathbf{s}$ is sampled once in **Key-Gen** and reused across decapsulations. We index queries by t and attach this index to every per-query quantity, while the fixed key material $\mathbf{s}, \mathbf{e}, \mathbf{A}, \mathbf{b}$ carries no index.

For query t , decryption decompresses $(c_{1,t}, c_{2,t})$ into $(\mathbf{u}'_t, \mathbf{v}'_t)$. Define the centered decompression errors

$$\Delta_{\mathbf{u},t} = (\mathbf{u}'_t - \mathbf{u}_t) \bmod^{\pm} q, \quad \Delta_{\mathbf{v},t} = (\mathbf{v}'_t - \mathbf{v}_t) \bmod^{\pm} q.$$

Substituting the encryption equations into $\mathbf{m}'_t = \mathbf{v}'_t - \langle \mathbf{u}'_t, \mathbf{s} \rangle$ gives, initially in $\mathcal{R}_q$,

$$\begin{aligned}\mathbf{m}'_t &= \langle \mathbf{b}, \mathbf{r}_t \rangle + \mathbf{e}_{2,t} + \lceil \mathbf{m}_t \rceil_{2 \rightarrow q} - \langle \mathbf{A}^\top \mathbf{r}_t + \mathbf{e}_{1,t} + \Delta_{\mathbf{u},t}, \mathbf{s} \rangle + \Delta_{\mathbf{v},t} \\ &= \lceil \mathbf{m}_t \rceil_{2 \rightarrow q} + \langle \mathbf{r}_t, \mathbf{e} \rangle - \langle \mathbf{e}_{1,t} + \Delta_{\mathbf{u},t}, \mathbf{s} \rangle + \mathbf{e}_{2,t} + \Delta_{\mathbf{v},t} \\ &= \lceil \mathbf{m}_t \rceil_{2 \rightarrow q} + \mathbf{w}_t,\end{aligned}$$

where we used $\langle \mathbf{s}, \mathbf{A}^\top \mathbf{r}_t \rangle = \langle \mathbf{A}\mathbf{s}, \mathbf{r}_t \rangle$ and $\mathbf{b} - \mathbf{A}\mathbf{s} = \mathbf{e}$. Thus, the fixed public-key error enters through $\langle \mathbf{r}_t, \mathbf{e} \rangle$, alongside the per-query encryption errors.

The decryption noise is therefore

$$\mathbf{w}_t = \langle \mathbf{r}_t, \mathbf{e} \rangle - \langle \mathbf{e}_{1,t} + \Delta_{\mathbf{u},t}, \mathbf{s} \rangle + \mathbf{e}_{2,t} + \Delta_{\mathbf{v},t}.$$

Decoding is correct whenever every centered coefficient satisfies $|w_{t,i}| < \lfloor q/4 \rfloor$, a condition that fails only with negligible probability. Although the decryption noise is discarded in normal operation, it is the quantity targeted by fault and side-channel attacks such as [PP21, RPJ⁺24].

For an authentic ciphertext, the attacker retains the message and hence knows $\mathbf{r}_t, \mathbf{e}_{1,t}, \mathbf{e}_{2,t}$. The decompression errors are also computable from the public ciphertext and encryption values. The only unknowns are therefore the long-term secret $\mathbf{s}$ and public-key error $\mathbf{e}$. Stacking their coefficient vectors as $[\mathbf{s}, \mathbf{e}] \in \mathbb{Z}^{2kn}$, we can write

$$\mathbf{w}_t = \mathbf{M}_t[\mathbf{s}, \mathbf{e}] + \mathbf{z}_t, \quad \mathbf{M}_t = \left[-\text{Neg}(\mathbf{e}_{1,t} + \Delta_{\mathbf{u},t}) \mid \text{Neg}(\mathbf{r}_t) \right], \quad \mathbf{z}_t = \mathbf{e}_{2,t} + \Delta_{\mathbf{v},t}, \quad (3)$$

where Neg of a length- k polynomial vector denotes the horizontal concatenation of its k negacyclic blocks.

Using centered representation ($\text{mod}^\pm q$), equation (3) can be treated over the integers, $\mathbf{M}_t \in \mathbb{Z}^{n \times 2kn}$ and $\mathbf{z}_t \in \mathbb{Z}^n$. On the correct-decoding event, the small-coefficient sum $\mathbf{M}_t[\mathbf{s}, \mathbf{e}] + \mathbf{z}_t$ lies in the centered decoding interval, so *no* modular wraparound occurs.

Remark 1. Keeping both $\mathbf{s}$ and $\mathbf{e}$ as unknowns is essential: it preserves a dense but small-coefficient integer system. Eliminating $\mathbf{e}$ through $\mathbf{b} = \mathbf{A}\mathbf{s} + \mathbf{e}$ would introduce the pseudorandom public matrix $\mathbf{A}$ into the design, causing the coefficient sums to wrap modulo q and turning the problem into a much harder modular inference problem.

3 Step 1: Extracting Per-Share Information

In the masked implementation the noisy message polynomial $\mathbf{m}' = \lceil \mathbf{m} \rceil_{2 \rightarrow q} + \mathbf{w}$, from which the decryption noise $\mathbf{w}$ is read off, is never formed in the clear: masked decryption keeps it as the two arithmetic shares $\mathbf{m}'_1, \mathbf{m}'_2$ of Eq. (2), with $\mathbf{m}'_1 + \mathbf{m}'_2 = \mathbf{m}' \text{ mod } q$, and processes each separately. Because the decapsulated message $\mathbf{m}$ is known, the pair determines the decryption noise: $\mathbf{w} = \mathbf{m}'_1 + \mathbf{m}'_2 - \lceil \mathbf{m} \rceil_{2 \rightarrow q} \text{ mod } q$. Algorithm 3 lists the leakage-relevant operations. We prefix masked variables with **masked-**, so that e.g. **masked-s** = $(\mathbf{s}_1, \mathbf{s}_2)$ are the two shares of the long-term secret.

Algorithm 3 Leakage-relevant part of masked decapsulation

```

1: function Refresh( $(\mathbf{s}_1, \mathbf{s}_2)$ )
2:    $\mathbf{h} \xleftarrow{\$} \mathcal{U}_q(\mathcal{R}_q^k)$ 
3:   return  $(\mathbf{s}_1 + \mathbf{h}, \mathbf{s}_2 - \mathbf{h})$  ▷ fresh sharing each query

4: function Masked-Decrypt( $c_1, c_2, (\mathbf{s}_1, \mathbf{s}_2)$ )
5:   masked-s  $\leftarrow$  Refresh(masked-s)
6:    $\mathbf{u}' \leftarrow \lceil c_1 \rceil_{D_u \rightarrow q}, \quad \mathbf{v}' \leftarrow \lceil c_2 \rceil_{D_v \rightarrow q}$ 
7:    $\mathbf{m}'_1 \leftarrow \mathbf{v}' - \langle \mathbf{u}', \mathbf{s}_1 \rangle$  ▷ arithmetic shares of  $\mathbf{m}'$ 
8:    $\mathbf{m}'_2 \leftarrow -\langle \mathbf{u}', \mathbf{s}_2 \rangle$ 
9:    $(\mathbf{m}_1, \mathbf{m}_2) \leftarrow \text{A2B-Compress}(\mathbf{m}'_1, \mathbf{m}'_2)$  ▷ loads  $\mathbf{m}'_1, \mathbf{m}'_2$  inside the trigger window
10:  return masked-m :=  $(\mathbf{m}_1, \mathbf{m}_2)$ 
```

Source (masked-poly.c)

```
for (i = 0; i < KYBER_SYMBYTES; i++) {
  for (j = 0; j < 8; j++) {
    KYBER_TRIG_HIGH();
    a1 = (a->polys[0].coeffs[8*i+j] - KYBER_Q/4 + KYBER_Q) % KYBER_Q;
    a2 = a->polys[1].coeffs[8*i+j];
    KYBER_TRIG_LOW();
    /* ... A2B conversion of (a1, a2) ... */ } }
```

Compiled code around the trigger interval

```
bl      trigger_high      @ window opens
ldrsh.w r4, [fp], #2      @ m'_1 = polys[0][8i+j] -> r4
ldrsh.w r3, [fp, #510]    @ m'_2 = polys[1][8i+j] -> r3
str     r3, [sp, #0]      @ m'_2 spilled to the stack
bl      trigger_low      @ window closes
bl      xorshift128      @ randomness for the A2B conversion
...
ldr     r3, [sp, #0]      @ m'_2 reloaded
...
addw    r4, r4, #2497     @ first use of m'_1 (then sdiv, mls)
```

Figure 2: Leak point inside `masked_poly_tomsg`: source (top) and compiled trigger window (bottom). `a->polys[0]` and `a->polys[1]` hold the arithmetic shares m'_1, m'_2 . Both loads are sign-extending (16 $\rightarrow$ 32 bits), which is why our leakage label is the Hamming weight of the sign-extended 32-bit word. m'_1 stays in the callee-saved `r4` until after the window, whereas m'_2 is loaded into the caller-saved `r3` and spilled to the stack before the window closes.

The `Refresh` routine re-randomizes the secret-key shares for every query. It is not part of the original implementation of [HKL⁺22]; we added it so that each decapsulation operates on a fresh sharing. Without it, the shares of m' are a deterministic function of the ciphertext, so an attacker could decapsulate the same ciphertext repeatedly and average the traces to suppress the measurement noise on the shares, making the original implementation easier to attack. With it, m'_1 and m'_2 are each uniform on $\mathbb{Z}_q$ in every query, as the leakage model of Section 4 assumes.

The `A2B-Compress` routine performs masked message decoding and returns Boolean shares satisfying $m = m_1 \oplus m_2$ [OSPG18, BDV21]. In the implementation, this operation is performed by `masked_poly_tomsg`. We target its initial loads because they expose both arithmetic-share coefficients within one execution (Figure 2). The trigger calls are our acquisition instrumentation. We model each load’s leakage by the Hamming weight of the word it writes to its destination register (`hw-sign`): the share’s 16-bit value sign-extended to 32 bits, matching the `ldrsh.w` loads of Figure 2.

3.1 Profiling with a Deep Neural Network

For each coefficient of m' , let $\alpha_s = \text{HW}(m'_s)$ denote the Hamming weight of share $s \in \{1, 2\}$ under the leakage model above, and let `seg` be the trace segment covering the two share loads (Figure 2). We estimate the two conditional distributions $p_{\alpha_1}(\cdot \mid \text{seg})$ and $p_{\alpha_2}(\cdot \mid \text{seg})$. Since all $n = 256$ coefficients run through the same instruction sequence, we pool their segments and train a single network rather than one per coefficient. It has a shared one-dimensional convolutional trunk and two softmax heads, one per share, trained jointly by minimizing the sum of their cross-entropies and selected on a held-out validation set. The two softmax outputs are our estimates $\hat{p}_{\alpha_1}(\cdot \mid \text{seg})$ and $\hat{p}_{\alpha_2}(\cdot \mid \text{seg})$, which Step 2 uses.

Architecture. The shared trunk is a one-dimensional CNN: six Conv–BatchNorm–ReLU blocks ($32 \rightarrow 64 \rightarrow 96 \rightarrow 128 \rightarrow 160 \rightarrow 160$ channels; kernel sizes 21/15/11/7/5/3) with max-pooling for downsampling and a final adaptive average-pool to 16 positions, followed by two fully connected layers (512 and 256 units) with batch normalization and dropout (0.35, 0.25). Each head is a linear softmax classifier over the 12 Hamming-weight classes of its share (Section 3.3).

3.2 Measurement Setup

Power traces were acquired with a NewAE ChipWhisperer-Husky Plus. The device under test is an STM32F4 (ARM Cortex-M4) on a CW308 UFO baseboard, clocked at 7.5 MHz from the capture board; the 12-bit ADC samples synchronously at $20\times$ the core clock (150 MS/s) with an input gain of 25 dB. In segmented-capture mode we record one segment per coefficient: it starts at the rising edge of the trigger of Figure 2 and collects 500 samples (25 clock cycles), independent of the falling edge. Profiling uses a separate set of 1000 decapsulations (256,000 segments) recorded on the same device, split into training, validation, and held-out test sets.

3.3 Experimental Results

How we measure leakage. Let $\text{seg}[\kappa]$ denote sample κ of a trace segment. For the Hamming-weight label α_s of share s , we define

$$\text{SNR}_s(\kappa) = \frac{\text{Var}(\mathbb{E}[\text{seg}[\kappa] \mid \alpha_s])}{\mathbb{E}[\text{Var}(\text{seg}[\kappa] \mid \alpha_s)]}. \quad (4)$$

This compares the variance of the label-dependent mean with the average within-class variance. An SNR of 1 means that these two variances are equal.

We evaluate both heads on the held-out test set $\mathcal{D}$ of triples (seg, a_1, a_2) , where a_s is the observed Hamming-weight label of share s . Writing $\hat{\mathbb{E}}_{\mathcal{D}}$ for the empirical average, we report

$$\begin{aligned} \text{ACC}_s &= \hat{\mathbb{E}}_{\mathcal{D}} [\mathbf{1}[a_s = \arg \max_a \hat{p}_{\alpha_s}(a \mid \text{seg})]], \\ \bar{p}_s &= \hat{\mathbb{E}}_{\mathcal{D}} [\hat{p}_{\alpha_s}(a_s \mid \text{seg})], \\ \text{CE}_s &= \hat{\mathbb{E}}_{\mathcal{D}} [-\log_2 \hat{p}_{\alpha_s}(a_s \mid \text{seg})], \\ \text{PI}_s &= H(\alpha_s) - \text{CE}_s. \end{aligned} \quad (5)$$

These are the top-1 accuracy, mean probability assigned to the true class, cross-entropy, and estimated perceived information. CE and PI are expressed in bits per coefficient segment. Since the attack uses the predicted distributions in full, we assess their quality primarily through CE and PI; accuracy and $\bar{p}_s$ provide complementary descriptive measures.

A trace-blind reference. For the reference values and label entropy, we use the uniform-share model with representatives in $\{0, \dots, q-1\}$, as in Section 4. Under this model, the Hamming weight has 12 classes, $0, \dots, 11$, with the largest probability, 0.236, at weight 6. A predictor that always outputs this class distribution has accuracy 0.236, mean true-class probability 0.171, and cross-entropy equal to the label entropy, 2.76 bits. Its perceived information is therefore zero. These are theoretical population values, shown in the last row of Table 2.

Results. Table 2 shows a substantial difference between the two heads. The second-share head achieves nearly perfect Hamming-weight classification and 2.72 bits of perceived information per coefficient segment, close to the label entropy of 2.76 bits. The first-share

head is considerably weaker, but its cross-entropy of 2.15 bits remains below the prior reference, corresponding to 0.61 bits of perceived information. Thus, its modest top-1 accuracy does not imply that its predicted distributions are uninformative. The SNR measurements show a similar asymmetry: the peak is 1.04 for the second share and 0.034 for the first, a ratio of about 30.

Table 2: Per-share classifier metrics on held-out profiling segments (**hw-sign** labels), together with the peak SNR. CE and PI are expressed in bits per coefficient segment. The final row gives the theoretical trace-blind reference under the uniform-share model.

Share	ACC	$\bar{p}$	CE (bits)	PI (bits)	peak SNR
m'_1 (polys[0])	0.358	0.270	2.15	0.61	0.034
m'_2 (polys[1])	0.992	0.988	0.037	2.72	1.04
Theoretical prior	0.236	0.171	2.76	0	—

Interpreting the leakage asymmetry. The two share coefficients have the same marginal distribution under the uniform-share model, but their handling in the compiled code differs. Figure 2 shows that, before `trigger_low`, the first coefficient is loaded into callee-saved `r4`, whereas the second is loaded into caller-saved `r3` and additionally spilled to the stack to preserve it across subsequent function calls. This extra memory operation provides a plausible explanation for the stronger second-share leakage.

The SNR curves in Figure 3 are consistent with this interpretation: the first share exhibits short, weak peaks, while the second exhibits two pronounced rises followed by a broader region of elevated SNR. The listing and curves do not isolate the contribution of individual instructions, so they support this explanation without establishing that the spill alone causes the full difference. The observed asymmetry may also depend on compiler choices and instrumentation. Section 4.1 evaluates the information remaining after the two predicted share distributions are combined.

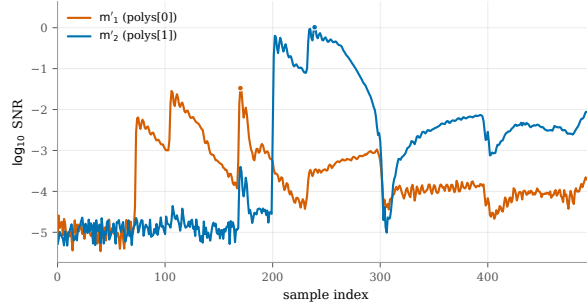


Figure 3: Per-sample Hamming-weight SNR for the two shares over the 500-sample segment, plotted as $\log_{10}$ SNR. Sampling is synchronous at 20 samples per core clock cycle. The markers indicate the peak SNR of each share.

4 Step 2: Combining Shares into a Decryption-Noise PMF

For each decapsulation t and coefficient i , Step 1 gives the two Hamming-weight estimates $\hat{p}_{\alpha_1}(\cdot \mid \text{seg})$ and $\hat{p}_{\alpha_2}(\cdot \mid \text{seg})$, where $\text{seg} = \text{seg}_{t,i}$. This section turns them into a PMF $L_{t,i}$ over the decryption-noise coefficient $w_{t,i}$, which the solver of Section 5 uses as a likelihood factor. The conversion has three steps: lift each Hamming-weight estimate to a distribution over share values, convolve the two value distributions modulo q , and remove

the known message offset. Until the last step we fix t and i , and write m'_1, m'_2, m', w for the corresponding coefficients and $m \in \{0, 1\}$ for the message bit.

From Hamming weights to share values. The heads constrain a share only through its Hamming weight. We identify each share with its representative in $\{0, \dots, q-1\}$, whose Hamming weight equals that of its sign-extended 32-bit word (Figure 2), and group the values into the classes

$$\mathcal{C}_\alpha = \{a \in \mathbb{Z}_q : \text{HW}(a) = \alpha\}, \quad \alpha = 0, \dots, 11.$$

Because **Refresh** makes each share uniform on $\mathbb{Z}_q$, a share is also uniform within its class, which gives the lift

$$\hat{p}_{m'_s}(a \mid \text{seg}) = \frac{\hat{p}_{\alpha_s}(\text{HW}(a) \mid \text{seg})}{|\mathcal{C}_{\text{HW}(a)}|}, \quad a \in \mathbb{Z}_q, s \in \{1, 2\}. \quad (6)$$

It adds no information beyond the head's output, and it is exact when the leakage depends on the share only through its Hamming weight, as the leakage model of Step 1 assumes.

Combining the shares. The two lifted distributions are combined by the length- q cyclic convolution

$$\hat{p}_{m'}(\nu \mid \text{seg}) = \sum_{a \in \mathbb{Z}_q} \hat{p}_{m'_1}(a \mid \text{seg}) \hat{p}_{m'_2}((\nu - a) \bmod^+ q \mid \text{seg}), \quad \nu \in \mathbb{Z}_q, \quad (7)$$

which we evaluate with the DFT as in the preliminaries.

Centering by the known message. The attacker generated the ciphertext, so it knows the message bit m and hence the offset $\lceil m \rceil_{2 \rightarrow q} \in \{0, 1665\}$. Since $w = m' - \lceil m \rceil_{2 \rightarrow q} \bmod q$, removing the offset is a cyclic shift. Restoring the indices, the factor that the solver uses for query t and coefficient i , with m the corresponding message bit, is

$$L_{t,i}(\omega) = \hat{p}_{m'}((\omega + \lceil m \rceil_{2 \rightarrow q}) \bmod^+ q \mid \text{seg}_{t,i}), \quad \omega \in \{-(q-1)/2, \dots, (q-1)/2\}. \quad (8)$$

4.1 How Informative Is the Hamming-Weight Pair?

A single share reveals nothing about the decryption noise. After **Refresh**, each share is uniform on $\mathbb{Z}_q$ and independent of w , and our held-out measurements agree: each share alone yields no perceived information. Information about w appears only when the two shares are combined. The attack is therefore second-order, which is consistent with first-order masking security.

How much does the pair reveal? In bits per decryption-noise coefficient:

unmasked, $\text{HW}(m')$ observed exactly:	$I(w; \text{HW}(m')) = 2.77$
masked, both share weights observed exactly:	$I(w; \alpha_1, \alpha_2) = 0.079$
masked, our trained heads:	PI = 0.025

The first line is a reference point: what an unmasked implementation, which holds m' itself, leaks if its Hamming weight is read exactly. The second assumes that both share weights are known exactly. The third is what our heads actually deliver, measured end-to-end on held-out traces for the factors $L_{t,i}$ of Eq. (8).

The masking penalty. From the first line to the second, about 97% of the information is lost, before any measurement noise. The reason is structural. Each Hamming weight is a coarse summary of a uniform value, and it is linked to $\mathbf{w}$ only through the modular sum of Eq. (7). The distribution over $\mathbf{w}$ therefore stays broad, which is why the solver must keep the whole distribution rather than a point estimate. From the second line to the third, information is lost to imperfect classification, mostly because of the weaker share (Section 3.3).

5 Step 3: Feeding the PMFs to a Belief-Propagation Solver

Step 2 turns every captured decapsulation $t \in \{1, \dots, T\}$ and coefficient $i \in \{0, \dots, n-1\}$ into a likelihood factor $L_{t,i}$ (Eq. (8)) over one decryption-noise coefficient. Each factor carries only about 0.025 bits (Section 4.1), but every decryption-noise coefficient is a known affine function of the same $2kn = 1536$ coefficients of the secret $\mathbf{s}$ and the public-key error $\mathbf{e}$, so the evidence of all $T \cdot n$ factors accumulates on the same unknowns.

5.1 The Model and the Posterior

Stack the secret and the public-key error into the single unknown $\mathbf{x} = [\mathbf{s}, \mathbf{e}]$, with coordinates x_ℓ for $\ell = 1, \dots, 2kn$. Each coordinate is drawn independently from χ_{η_1} , with $\eta_1 = 2$ for KYBER768 (Table 1), so it lies in

$$\mathcal{S} = \{-2, -1, 0, 1, 2\}, \text{ with prior probability } \pi = \left(\frac{1}{16}, \frac{4}{16}, \frac{6}{16}, \frac{4}{16}, \frac{1}{16}\right). \quad (9)$$

By Eq. (3), the decryption noise of query t is $\mathbf{w}_t = \mathbf{M}_t \mathbf{x} + \mathbf{z}_t$ over the integers, where $\mathbf{M}_t \in \mathbb{Z}^{n \times 2kn}$ and $\mathbf{z}_t \in \mathbb{Z}^n$ are known to the attacker. We write $M_{i,\ell}$ and $z_{t,i}$ for their entries, leaving t implicit in $M_{i,\ell}$. The blocks of $\mathbf{M}_t$ are negacyclic matrices, so its rows are dense: *every* decryption-noise coefficient depends on *all* $2kn$ unknowns.

Each factor $L_{t,i}$ is a soft measurement of one entry, $w_{t,i} = (\mathbf{M}_t \mathbf{x} + \mathbf{z}_t)_i$. Because each share is uniform on $\mathbb{Z}_q$, $L_{t,i}$ is proportional to the likelihood of $w_{t,i}$: it contains no prior on the decryption noise, which enters only through π . Assuming that different segments are independent given $\mathbf{x}$, the posterior over the key is

$$p_{\mathbf{x}}(\mathbf{x} \mid \{\text{seg}_{t,i}\}) \propto \underbrace{\prod_{\ell=1}^{2kn} \pi(x_\ell)}_{\text{prior}} \prod_{t=1}^T \prod_{i=0}^{n-1} \underbrace{L_{t,i}((\mathbf{M}_t \mathbf{x} + \mathbf{z}_t)_i)}_{f_{t,i}(\mathbf{x})}. \quad (10)$$

Since the factors are the network’s estimates, this is the posterior under our leakage model, not the exact one. Its factor graph (Figure 4) has one variable per key coordinate, one prior factor per variable, and one observation factor $f_{t,i}$ per pair (t, i) . Every observation factor connects to all $2kn$ variables. In soft analytical side-channel attacks, by contrast, each factor involves only the few variables of one operation [VGS14]. This density makes the exact BP messages intractable; Section 5.2 shows how to approximate them.

5.2 The Belief-Propagation Solver

Belief propagation (BP) is a message-passing algorithm in which each key coordinate x_ℓ carries a *belief* B_ℓ about its value: a PMF over $\mathcal{S}$ that starts at the prior π . In every iteration, or *sweep*, each observation factor $f_{t,i}$ sends every coordinate a *message* $m_{f_{t,i} \rightarrow x_\ell}(a)$, which says how well the hypothesis $x_\ell = a$ explains the measured PMF $L_{t,i}$, given what is currently believed about the other coordinates. Each belief then combines the prior with all the messages it receives. Evidence too weak to decide anything within

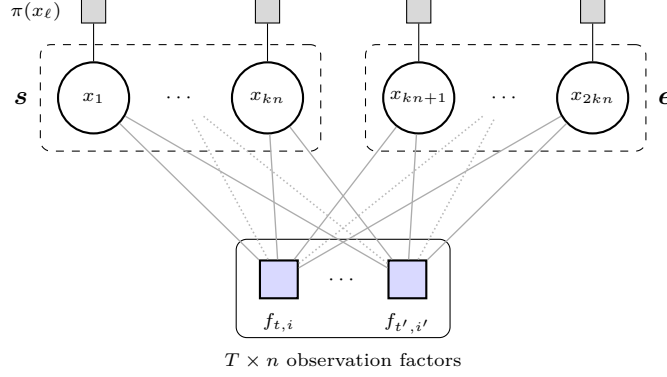


Figure 4: Factor graph of the posterior (10). Circles are the $2kn$ key coordinates with value $x_\ell \in \mathcal{S}$. The first kn form s and the rest form e . Grey squares are the prior factors $\pi(x_\ell)$. Each of the $T \times n$ blue squares is an observation factor $f_{t,i}(\mathbf{x}) = L_{t,i}((M_t \mathbf{x} + \mathbf{z}_t)_i)$, connected to every variable.

one factor thus accumulates over the $T \cdot n$ factors, and sharper beliefs make the next sweep's messages sharper in turn.

What makes our graph unusual is that every factor involves all $2kn$ coordinates (Figure 4). We therefore build the message in four parts: we write it as an average over the other coordinates (A), replace their joint effect by a Gaussian (B), keep a factor from hearing its own evidence back (C), and compute all of this without storing a message per edge (D). Algorithm 4 summarizes the result. Below, ℓ is the coordinate that receives a message and ℓ' ranges over the others.

A. The message is an average over the other coordinates. Fix a factor $f_{t,i}$ and a hypothesis $x_\ell = a$. The exact sum-product message averages the measured PMF over the coordinates that are still unknown:

$$m_{f_{t,i} \rightarrow x_\ell}(a) = \mathbb{E} \left[L_{t,i} \left(z_{t,i} + M_{i,\ell} a + \sum_{\ell' \neq \ell} M_{i,\ell'} x_{\ell'} \right) \right], \quad (11)$$

where each $x_{\ell'}$ is drawn independently from $m_{x_{\ell'} \rightarrow f_{t,i}}$, the distribution that the rest of the graph currently assigns to it (Part C). The argument of $L_{t,i}$ is the decryption-noise coefficient $w_{t,i}$ that these values imply. The average runs over the $5^{2kn-1} = 5^{1535}$ joint values of the other coordinates, so it cannot be computed as written.

B. The other coordinates act as Gaussian interference. The sum over $\ell' \neq \ell$ in Eq. (11) has 1535 independent, bounded terms, so by the central limit theorem it is close to Gaussian, and only its mean and variance matter:

$$\mu_{t,i}^{(\ell)} = z_{t,i} + \sum_{\ell' \neq \ell} M_{i,\ell'} \mu_{\ell' \rightarrow f}, \quad (12)$$

$$v_{t,i}^{(\ell)} = \sum_{\ell' \neq \ell} M_{i,\ell'}^2 v_{\ell' \rightarrow f}, \quad (13)$$

where $\mu_{\ell' \rightarrow f}$ and $v_{\ell' \rightarrow f}$ are the mean and variance of $m_{x_{\ell'} \rightarrow f_{t,i}}$. The 1535-dimensional average then collapses to a one-dimensional one,

$$m_{f_{t,i} \rightarrow x_\ell}(a) \approx \sum_{y \in \mathbb{Z}} L_{t,i}(y) \mathcal{N}(y; \mu_{t,i}^{(\ell)} + M_{i,\ell} a, v_{t,i}^{(\ell)}), \quad (14)$$

with the Gaussian read on the integers, since the decryption noise is an integer. In words: smear the measured PMF by the uncertainty that the other coordinates still leave, and read it off where the hypothesis $x_\ell = a$ puts the decryption noise. Only this interference is Gaussian; the PMF itself is kept in full, so the skew and multiple peaks that separate key hypotheses survive.

The width of the smearing drives the whole attack. In the first sweep every belief equals the prior, so $v_{t,i}^{(\ell)} \approx 2.2 \cdot 10^3$, a standard deviation of about 47, and each message is nearly flat. As the beliefs sharpen, $v_{t,i}^{(\ell)}$ shrinks and the messages become decisive.

C. A factor must not hear its own evidence. Which distribution should $x_{\ell'}$ follow in Eq. (11)? Its belief $B_{\ell'}$ is the obvious choice, but $B_{\ell'}$ already contains the message that $f_{t,i}$ itself sent in the previous sweep. That message moved the belief slightly, in the direction that brings the decryption noise implied by $x_{\ell'}$ closer to what $L_{t,i}$ favours. Each move is tiny, but the moves of all 1535 coordinates add up in the sum of Eq. (12): fed with the beliefs, the factor would get its own estimate of the decryption noise back, as if the other coordinates had predicted it (Part D quantifies this). This *echo* counts, as the same observation twice and makes the beliefs overconfident. BP therefore uses the *cavity* distribution, the belief with this factor's contribution divided out:

$$m_{x_{\ell'} \rightarrow f_{t,i}} \propto \frac{B_{\ell'}}{m_{f_{t,i} \rightarrow x_{\ell'}}}, \quad B_{\ell'} \propto \pi \prod_f m_{f \rightarrow x_{\ell'}}. \quad (15)$$

For the same reason, the sums in Eqs. (12) and (13) skip $\ell' = \ell$: the receiving coordinate is fixed to a , so it only shifts the Gaussian by $M_{i,\ell} a$.

D. Cavities without storing every message. Eq. (15) needs every factor's latest message to every coordinate, $T \cdot n \cdot 2kn \cdot |S|$ numbers, about $4.3 \cdot 10^9$ at $T = 2200$. Approximate message passing on dense graphs (AMP [DMM09], GAMP [Ran11]) avoids this storage. It computes everything from the beliefs, which all factors share, and then removes each factor's own contribution, which two numbers per factor describe.

What the beliefs predict. Let $\mu_{\ell'}$ and $\sigma_{\ell'}^2$ be the mean and variance of $B_{\ell'}$. Over all coordinates, the beliefs predict the decryption noise $w_{t,i}$ to have mean and variance

$$\mu_{t,i}^{\text{tot}} = z_{t,i} + \sum_{\ell'} M_{i,\ell'} \mu_{\ell'}, \quad V_{t,i} = \sum_{\ell'} M_{i,\ell'}^2 \sigma_{\ell'}^2. \quad (16)$$

What the factor does to the prediction. Let $c_{t,i}$ and $\bar{V}_{t,i} \approx V_{t,i}$ be the predicted mean and variance without the factor's own echo (Eq. (19) below). Combining the prediction $\mathcal{N}(c_{t,i}, \bar{V}_{t,i})$ with the measured PMF gives the factor's own posterior on the decryption noise, $P_{t,i}(y) \propto L_{t,i}(y) \mathcal{N}(y; c_{t,i}, \bar{V}_{t,i})$, with mean $\hat{w}_{t,i}$ and variance $\tau_{t,i}$. Two numbers summarize how the factor changes the prediction:

$$\psi'_{t,i} = \frac{\hat{w}_{t,i} - c_{t,i}}{\bar{V}_{t,i}}, \quad \psi''_{t,i} = \frac{\tau_{t,i} - \bar{V}_{t,i}}{\bar{V}_{t,i}^2}. \quad (17)$$

Thus ψ' is the factor's pull on the mean of the decryption noise and ψ'' its change to the variance, both scaled by $\bar{V}_{t,i}$; a flat PMF gives $\psi' = \psi'' = 0$. They are the first two derivatives of $\log \sum_y L_{t,i}(y) \mathcal{N}(y; c, \bar{V}_{t,i})$ with respect to c , at $c = c_{t,i}$: the output function of GAMP and its derivative [Ran11].

The size of the echo. By Eq. (14), the message of $f_{t,i}$ to $x_{\ell'}$ is this function read at a mean shifted by $M_{i,\ell'} x_{\ell'}$. Near the current beliefs, its logarithm is therefore approximately quadratic in $x_{\ell'}$, with slope $M_{i,\ell'} \psi'_{t,i}$ and curvature $M_{i,\ell'}^2 \psi''_{t,i}$, and it has moved the mean of $B_{\ell'}$ by about $M_{i,\ell'} \sigma_{\ell'}^2 \psi'_{t,i}$. One such move is tiny, but in the predicted decryption noise

Algorithm 4 The BP solver. One pass of the outer loop is a sweep.

Require: PMFs $L_{t,i}$; M_t , $\mathbf{z}_t$; prior π ; public key $(\mathbf{A}, \mathbf{b})$; damping γ

- 1: $B_\ell \leftarrow \pi$ for every ℓ ; $\psi'_{t,i}, \psi''_{t,i} \leftarrow 0$ for every (t, i)
- 2: **for** sweep = 1, ..., 100 **do**
- 3: $\mu_\ell, \sigma_\ell^2 \leftarrow$ mean and variance of B_ℓ for every ℓ
- 4: **for** every factor (t, i) **do**
- 5: $\mu_{t,i}^{\text{tot}}, V_{t,i} \leftarrow$ totals over all coordinates ▷ Eq. (16)
- 6: $c_{t,i}, \bar{V}_{t,i} \leftarrow$ factor cavity moments from $\psi'_{t,i}, \psi''_{t,i}$ ▷ Eq. (19)
- 7: **for** every coordinate ℓ **do**
- 8: $\mu_{t,i}^{(\ell)}, v_{t,i}^{(\ell)} \leftarrow$ cavity moments ▷ Eqs. (20)–(21)
- 9: $m_{f_{t,i} \rightarrow x_\ell}(a)$ for every $a \in \mathcal{S}$ ▷ Eq. (14)
- 10: update $\psi'_{t,i}, \psi''_{t,i}$ from $P_{t,i}$, damped by γ ▷ Eq. (17)
- 11: update every belief B_ℓ ▷ Eq. (22)
- 12: $\hat{x}_\ell \leftarrow \arg \max_a B_\ell(a)$; split $\hat{x}$ into $\hat{s}, \hat{e}$
- 13: **if** $\mathbf{b} = \mathbf{A}\hat{s} + \hat{e}$ **then return** $(\hat{s}, \hat{e})$
- 14: **return** $(\hat{s}, \hat{e})$ ▷ best estimate; key not confirmed

the moves of all coordinates add up to $\sum_{\ell'} M_{i,\ell'} \cdot M_{i,\ell'} \sigma_{\ell'}^2 \psi'_{t,i} = V_{t,i} \psi'_{t,i} \approx \hat{w}_{t,i} - c_{t,i}$, the factor's whole pull. Dividing the message out of $B_{\ell'}$ undoes the move, to first order:

$$\mu_{\ell' \rightarrow f} \approx \mu_{\ell'} - M_{i,\ell'} \sigma_{\ell'}^2 \psi'_{t,i}, \quad v_{\ell' \rightarrow f} \approx \sigma_{\ell'}^2 - M_{i,\ell'}^2 \sigma_{\ell'}^4 \psi''_{t,i}. \quad (18)$$

Summed over all coordinates, these corrections give the factor's cavity moments

$$c_{t,i} = \mu_{t,i}^{\text{tot}} - \psi'_{t,i} V_{t,i}, \quad \bar{V}_{t,i} = V_{t,i} - \psi''_{t,i} \sum_{\ell'} M_{i,\ell'}^4 \sigma_{\ell'}^4, \quad (19)$$

and summed over all coordinates but the receiver, they turn Eqs. (12) and (13) into

$$\mu_{t,i}^{(\ell)} = \underbrace{\mu_{t,i}^{\text{tot}}}_{\text{all coordinates}} - \underbrace{M_{i,\ell} \mu_\ell}_{\text{receiver}} - \underbrace{\psi'_{t,i} (V_{t,i} - M_{i,\ell}^2 \sigma_\ell^2)}_{\text{echo}}, \quad (20)$$

$$v_{t,i}^{(\ell)} = V_{t,i} - M_{i,\ell}^2 \sigma_\ell^2 - \psi''_{t,i} \left(\sum_{\ell'} M_{i,\ell'}^4 \sigma_{\ell'}^4 - M_{i,\ell}^4 \sigma_\ell^4 \right). \quad (21)$$

The echo term is the Onsager reaction term of AMP. Its counterpart in the variance is much smaller while many coordinates are uncertain, because fourth powers do not add up the way the mean moves do; we keep it nonetheless. The echo is negligible while the beliefs are broad, but it dominates once they sharpen (Section 5.3).

Order of computation. Eqs. (17) and (19) depend on each other, so each sweep forms $c_{t,i}$ and $\bar{V}_{t,i}$ with the derivatives of the previous sweep and then updates them, damped like the beliefs, so that they match the mix of past messages that the beliefs contain. In the first sweep, $\psi' = \psi'' = 0$. Checked against exact division of each factor's message out of saved beliefs, Eq. (18) reproduces the exact cavity messages to a median relative error of 0.6% at the partial-recovery point, whereas omitting it errs by 61%.

Belief update and stopping rule. A sweep ends by adding every factor's log-message, together with the log-prior, onto the beliefs, damped for stability [Hes02, Pre05]:

$$\log \tilde{B}_\ell(a) = \log \pi(a) + \sum_{t=1}^T \sum_{i=0}^{n-1} \log m_{f_{t,i} \rightarrow x_\ell}(a),$$

$$\log B_\ell \leftarrow \gamma \log \tilde{B}_\ell + (1 - \gamma) \log B_\ell, \quad \gamma = 0.2, \quad (22)$$

followed by normalization over $\mathcal{S}$. We then take the most likely value of every coordinate, $\hat{x}_\ell = \arg \max_a B_\ell(a)$, split the estimate into $\hat{s}$ and $\hat{e}$, and test it against the public key, $\mathbf{b} = \mathbf{A}\hat{s} + \hat{e}$. The test needs no knowledge of the key, and a wrong estimate passes it only with negligible probability, so the attacker knows when recovery is complete. We stop at the first sweep that passes, or after 100 sweeps.

5.3 Experimental Results

Key recovery on the device. Figure 5a shows the accuracy of BP on the real capture versus the number of traces T , over all 1536 coordinates and over the nonzero ones. Accuracy grows steadily with T . At $T = 2000$, BP stalls at 70% nonzero accuracy. From $T = 2200$ on, it recovers the full key; at $T = 2200$, this takes about two hours on a laptop CPU (Intel Core i7-1280P, 16 GB of RAM, no GPU). Removing the echo (Parts C and D of Section 5.2) is essential: with the echo left in, BP stalls at 73% nonzero accuracy at $T = 2800$ and recovers the key only at $T = 3500$.

Simulation model. To relate the trace count to the leakage, we simulate a masked and an unmasked implementation under a Hamming-weight leakage model. We keep the encryption randomness and messages of the capture, so every $\mathbf{M}_t$ and $\mathbf{z}_t$ is genuine, and draw a fresh key from χ_2 , which fixes every noisy message coefficient $\mathbf{m}'$. With a fresh uniform sharing $\mathbf{m}' = \mathbf{m}'_1 + \mathbf{m}'_2 \bmod q$ and independent measurement noise $\varepsilon, \varepsilon_s \sim \mathcal{N}(0, \sigma^2)$, the attacker observes

$$\tilde{\alpha}_s = \text{HW}(\mathbf{m}'_s) + \varepsilon_s, \quad s \in \{1, 2\} \quad (\text{masked}), \quad \tilde{\alpha} = \text{HW}(\mathbf{m}') + \varepsilon \quad (\text{unmasked}). \quad (23)$$

It uses the exact Hamming-weight posteriors of this model in place of the network; this idealized attacker has no estimation error. From there on, Eqs. (6) to (8) (without the convolution for the unmasked target) and the solver are unchanged. For each σ we compute the information $I = I(\mathbf{w}; \tilde{\alpha}_1, \tilde{\alpha}_2 \mid \mathbf{m})$, or $I(\mathbf{w}; \tilde{\alpha} \mid \mathbf{m})$, exactly, and bisect for the smallest T that recovers all coefficients within 100 sweeps.

Traces versus information. Figure 5b plots these thresholds against I . Each trace yields $n = 256$ observations, independent given the key, so T traces reveal at most TnI bits about it. The key has $H = 2kn H(\chi_2) \approx 3119$ bits of entropy. Information-theoretically, the public key alone already determines it, but extracting that information means solving the MLWE instance, the task of the lattice step of Section 6. Our solver uses the public key only to confirm a candidate (Section 5.2). For such leakage-only recovery, the traces must supply the H bits themselves, so $T \geq H/(nI)$, and the thresholds follow this bound up to a nearly constant factor:

$$T \approx c \frac{H}{nI}, \quad (24)$$

with $c = 3.36 \pm 0.08$ for the unmasked target over a twelvefold range of I ($2 \leq \sigma \leq 8$), and c between 3.7 and 3.9 for the masked target ($\sigma \geq 0.7$); c grows at higher information (lower σ). At equal measurement noise, masking raises the trace count 31 to 50 times, close to the 35- to 52-fold drop in information: masking costs traces mainly through the information it removes. The device fits the same rule: with the perceived information 0.025 of Section 4.1, its 2200 traces give $c \approx 4.5$, about 20% above the simulated masked target at the same information. This gap combines the network’s estimation error, the device’s unequal leakage of the two shares, which the simulation does not model, and key-to-key variation.

5.4 The Least-Squares Baseline and Why It Fails

Equation (3) is a linear system in the key: $\mathbf{M}_t$ and $\mathbf{z}_t$ are known, and each decryption-noise coefficient $w_{t,i}$ is observed only through the PMF $L_{t,i}$ of Eq. (8). The natural first attempt is ordinary least squares, the standard way to invert such a system and the optimal one when each observation is the true value plus Gaussian measurement noise [SL03]. Least squares needs a number per row, however, not a distribution. Drawing each $w_{t,i}$ from $L_{t,i}$ would make the estimate a random vector with an intractable distribution, but the

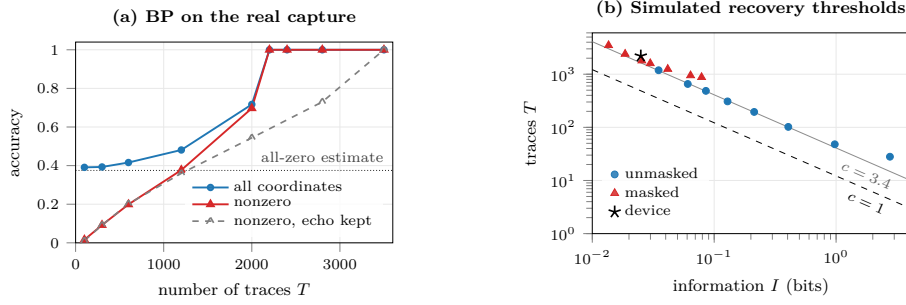


Figure 5: (a) Accuracy of BP on the real capture versus the number of traces T . Dashed: nonzero accuracy with the echo left in. (b) Recovery thresholds of the simulated attacker of Eq. (23) versus the information per decryption-noise coefficient, with the leakage-only bound $c = 1$ and the fit $c = 3.4$ of Eq. (24). The device point (star) uses the perceived information.

estimate is linear in the draws, so its expectation is simply the least-squares solution for the means $\bar{w}_{t,i} = \sum_{\omega} \omega L_{t,i}(\omega)$:

$$\hat{\mathbf{x}} = \left(\sum_t \mathbf{M}_t^\top \mathbf{M}_t \right)^{-1} \sum_t \mathbf{M}_t^\top (\bar{\mathbf{w}}_t - \mathbf{z}_t), \quad \bar{\mathbf{w}}_t = (\bar{w}_{t,0}, \dots, \bar{w}_{t,n-1}), \quad (25)$$

which we round coordinatewise to $\mathcal{S}$. All $L_{t,i}$ are about equally broad, so weighting the rows by their inverse variance would change little.

This estimator fails, for two reasons. The minor one is that it ignores the prior: $\hat{\mathbf{x}}$ ranges over all real vectors, and χ_{η_1} enters only through the final rounding. The decisive one is that the prerequisite behind its optimality fails: each observation should be the true value plus Gaussian measurement noise, so that $L_{t,i}$ is a Gaussian peak near $w_{t,i}$. Masking instead leaves each $L_{t,i}$ almost flat over $\mathbb{Z}_q$ (Section 4.1). Its information lies in how it ranks the few hundred values that the decryption noise can actually take, and a mean over all of $\mathbb{Z}_q$ averages this away, so the means carry almost no information about the key. In our experiments, the accuracy of $\hat{\mathbf{x}}$ stays at that of the all-zero key. Our BP solver avoids the problem: it makes its Gaussian approximation only for the interference, a sum over many coordinates, and keeps each PMF intact (Section 5.2). The greedy solver examined next avoids the means altogether, yet masking defeats it too.

5.5 The Greedy Inequality Baseline and Why It Fails

The state-of-the-art solver for Hamming-weight leakage of $\mathbf{m}'$ is the greedy inequality solver of Ravi et al. [RPJ⁺24]. It recovers the key of unprotected KYBER768 from pqm4 on a Cortex-M4 from a few thousand traces, and [RPJ⁺24, Sec. 8] evaluates it, in simulation, on the masked implementation [HKL⁺22] that we also target. Where least squares reduces each PMF to its mean, this solver reduces each observation to a hard interval.

From Hamming weights to intervals. Following [RPJ⁺24], the attacker uses only the coefficients whose message bit is zero, where the two shares sum to the decryption noise itself. Let $\mathcal{W}(\alpha_1, \alpha_2)$ be the set of decryption-noise values observed in a profiling set together with share weights (α_1, α_2) . The attacker tabulates its extremes,

$$\theta(\alpha_1, \alpha_2) = \min \mathcal{W}(\alpha_1, \alpha_2), \quad \Omega(\alpha_1, \alpha_2) = \max \mathcal{W}(\alpha_1, \alpha_2), \quad (26)$$

and turns each such observation (t, i) into a pair of linear inequalities,

$$\theta_{t,i} \leq y_{t,i}(\mathbf{x}) \leq \Omega_{t,i}, \quad y_{t,i}(\mathbf{x}) = (\mathbf{M}_t \mathbf{x} + \mathbf{z}_t)_i, \quad (27)$$

with $\theta_{t,i}$ and $\Omega_{t,i}$ read from Eq. (26) at the observed weights. Here $y_{t,i}(\mathbf{x})$ is the decryption noise that a candidate key $\mathbf{x}$ implies.

The greedy search. The solver looks for a key that violates the intervals as little as possible, as measured by

$$V(\mathbf{x}) = \sum_{t,i} \left([\theta_{t,i} - y_{t,i}(\mathbf{x})]_+ + [y_{t,i}(\mathbf{x}) - \Omega_{t,i}]_+ \right), \quad [a]_+ = \max(a, 0). \quad (28)$$

Following [RPJ⁺24, Algorithms 7–8], the search starts from $\mathbf{x} = \mathbf{0}$. In iteration k , it finds for every coordinate the step $x_\ell \leftarrow x_\ell + v$, $v \in \{\pm 1, \pm 2\}$, that decreases V the most, and applies these steps to the $\max(1, \lceil \alpha_0 \gamma^k \rceil)$ coordinates with the largest decrease, with $\alpha_0 = 256$ and $\gamma = 0.88$; it stops when no step decreases V . We re-implement it on the model of Section 5.1. As a check, with tight synthetic intervals $w_{t,i} \pm 8$ it recovers the full key from 40 traces in about a second.

Why it works without masking. Greedy needs intervals that are *correct*, $w_{t,i} \in [\theta_{t,i}, \Omega_{t,i}]$, so that the true key satisfies them, and *informative*, so that wrong keys do not. On an unprotected implementation both hold. For a zero message bit, the device stores the decryption noise itself as a signed 16-bit value, and a small negative value has a large Hamming weight in two’s complement, so the weight reveals the sign of the decryption noise [RPJ⁺24, Fig. 4a]. Seven of the 17 weight classes then give an interval on one side of zero [RPJ⁺24, Table 2]. Such an interval rules out every candidate key whose implied decryption noise $y_{t,i}(\mathbf{x})$ has the wrong sign, and each violation pulls the search toward the key.

Why it fails with masking. With masking and a zero message bit, a weight pair is compatible with a decryption-noise value ω whenever

$$\omega \bmod q \in \mathcal{C}_{\alpha_1} + \mathcal{C}_{\alpha_2} = \{a + b \bmod q : a \in \mathcal{C}_{\alpha_1}, b \in \mathcal{C}_{\alpha_2}\}.$$

Each class $\mathcal{C}_\alpha$ is scattered over all of $\mathbb{Z}_q$, so for all but the rarest weight pairs this sumset contains decryption-noise values of both signs: an interval that contains the true decryption noise contains zero as well. We therefore gave greedy the best case, a perfect Hamming-weight oracle, and it still failed: in a simulation of 3000 KYBER768 decryptions it ends at accuracy 0.31, below the 0.38 of the all-zero key. Only 0.13% of the intervals exclude zero, which leaves the solver two bad options.

- *Tight intervals exclude the key.* With tables built from about 10^5 observations, as in [RPJ⁺24], rare extreme decryption-noise values fall outside the tabulated ranges, and the true key violates the system more than the all-zero key: $V(\mathbf{x}_{\text{true}}) \approx 2.2 V(\mathbf{0})$.
- *Loose intervals admit wrong keys.* Intervals wide enough to contain the truth exclude almost nothing, so the all-zero key and many other wrong keys satisfy nearly the whole system.

This agrees with the masked simulation of [RPJ⁺24, Fig. 7], which needs about 6×10^4 ciphertexts even with perfect weights, more than an order of magnitude beyond our budget.

6 Step 4: The Lattice Step and Why It Does Not Help

Lattice-assisted side-channel attacks usually do not ask the statistical solver to finish alone. The solver narrows or fixes key coordinates and hands them to a lattice attack as hints, which completes the recovery [HMS⁺23, DDGR20]. We evaluate this cascade on the output of our BP solver. The result is negative: below the recovery threshold the BP marginals are well calibrated but too diffuse. They lower the estimated block size of the primal attack from 624 to about 505 at best, and it becomes small only one or two sweeps before BP recovers the key on its own. The lattice step therefore does not reduce the number of traces.

6.1 Lattice Background

A lattice is the set of all integer combinations of linearly independent basis vectors $\mathbf{b}_1, \dots, \mathbf{b}_r \in \mathbb{R}^{N'}$. An LWE secret can be encoded as an unusually short vector planted in such a lattice, and recovering the secret amounts to finding that vector. The tool for this is lattice reduction: LLL [LHWLL82] or, for stronger reduction, BKZ with a tunable *block size* β [SE91, CN11]. A larger β produces a better basis but costs exponentially more, so β is the natural single-number measure of attack cost. We use the primal attack throughout. The quality of a BKZ- β -reduced basis is summarized by the root-Hermite factor [GN08, ADPS16]

$$\delta_\beta \approx ((\pi\beta)^{1/\beta} \beta / 2\pi e)^{1/(2\beta-2)},$$

and under the geometric-series assumption [Sch03] it fixes the shape of the whole reduced basis. This turns “does BKZ- β recover the planted vector?” into an inequality in β (Eq. (29) below). We report the smallest such β as our cost measure. For reference, the core-SVP model [ADPS16, BDGL16] puts the classical cost near $2^{0.292\beta}$: $\beta \approx 150$ costs about 2^{44} operations, whereas β of several hundred is firmly out of reach.

6.2 From BP Marginals to Lattice Hints

The lattice attacks the public key, $\mathbf{b} = \mathbf{A}\mathbf{s} + \mathbf{e} \bmod q$, whose unknowns are the same $2kn = 1536$ coordinates x_ℓ of $\mathbf{x} = [\mathbf{s}, \mathbf{e}]$ that BP estimates. From each belief B_ℓ , after a given sweep, we take its most likely value $\hat{x}_\ell$, its confidence $c_\ell = \max_a B_\ell(a)$, and its mean μ_ℓ and standard deviation σ_ℓ . These enter the lattice as the two kinds of hints of Dachman-Soled et al. [DDGR20].

- *Perfect hints.* Every coordinate with $c_\ell \geq \tau$, or with $\sigma_\ell < 10^{-3}$, is fixed to $\hat{x}_\ell$. Fixing an error coordinate removes the unknown from its equation, and fixing a secret coordinate removes a column of $\mathbf{A}$; either way the lattice dimension drops by one. The fixed values must all be right. The attacker’s own estimate of that probability is $\prod c_\ell$ over the fixed set, which is meaningful only because the marginals are calibrated.
- *Approximate hints.* Every remaining coordinate is centred on μ_ℓ and scaled by $1/\sigma_\ell$. This makes the planted vector isotropic and multiplies the lattice volume by $\prod_\ell 1/\sigma_\ell$: every coordinate that BP knows better than the prior helps, even if it is far from certain.

Let $m = kn$ be the number of public-key equations, and m_f and n_f the numbers of error and secret coordinates left free. With the Kannan embedding [Kan87], the lattice has dimension $d = m_f + n_f + 1$ and volume V , and the planted vector is $(\mathbf{x}', 1)$ with $x'_\ell = (x_\ell - \mu_\ell)/\sigma_\ell$, where

$$\log V = (m_f + \min(m - m_f, n_f)) \log q - \sum_{\ell \text{ free}} \log \sigma_\ell.$$

Following the 2016 primal-uSVP estimate [ADPS16, APS15], BKZ- β finds the planted vector once

$$\sqrt{\|\mathbf{x}'\|^2 + 1} \sqrt{\beta/d} \leq \delta_\beta^{2\beta-d} V^{1/d}, \quad (29)$$

and we report the smallest such β . We minimize it over five thresholds τ between 0.9 and 0.9999, keeping only those whose fixed set is error-free and has a predicted success probability of at least 0.5, a choice that favours the attacker. We evaluate $\|\mathbf{x}'\|$ on the true key, so overconfident marginals are penalized rather than rewarded; for calibrated ones, $\|\mathbf{x}'\|^2/d \approx 1$. As checks, prior-only marginals give $\beta = 624$, the usual core-SVP level of KYBER768, and a fully recovered key gives an empty lattice.

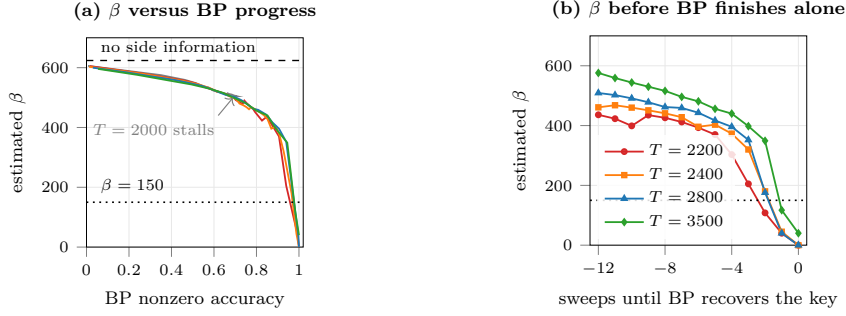


Figure 6: Estimated primal block size β with BP’s marginals as hints, recomputed after every sweep. (a) All runs follow the same curve; at $T = 2000$ (grey), where BP stalls, β ends at 505. Colours as in (b). (b) Aligned at the sweep where BP recovers the key by itself, β falls below 150 only one or two sweeps earlier.

Table 3: Estimated primal block size with BP’s marginals as hints, after the given sweep. Perfect hints use the best safe threshold τ . $\|\mathbf{x}'\|^2/d \approx 1$ indicates calibrated marginals, and values below 1 cautious ones. Without side information, $\beta = 624$.

T	sweep	nonzero acc.	$\ \mathbf{x}'\ ^2/d$	β , approximate	β , with perfect	coordinates fixed
100	30	0.016	0.98	621	621	0
600	30	0.199	0.96	606	606	0
1200	30	0.377	0.98	582	581	1
2000	100	0.697	1.04	510	505	9
2200	50	0.892	0.71	406	393	34
2800	19	0.972	0.17	299	176	381
2800	20	0.997	0.03	148	40	958

6.3 How Much the Hints Help

Table 3 and Figure 6 give the estimate for the BP runs of Section 5.3. Below the recovery threshold, the hints help little. Where BP stalls, at $T = 2000$, they lower β from 624 to 505, still far out of practical reach. Almost all of this gain comes from the approximate hints: very few coordinates are confident enough to be fixed safely, and even an oracle that knew which confident coordinates are correct could fix only about two dozen. As BP converges, β does fall, but too late: it drops below 150 only one or two sweeps before BP recovers the key by itself. The lattice step therefore never arrives in time to reduce the number of traces.

7 Conclusion

We recovered the long-term key of a first-order masked KYBER768 implementation on an ARM Cortex-M4 from about 2200 decapsulation traces with authentic ciphertexts. Masking leaves about 97% less information per decryption-noise coefficient than an unmasked Hamming weight, yet this weak leakage accumulates into full recovery when it is kept probabilistic: belief propagation with Onsager-corrected cavities succeeds where least squares and the greedy inequality solver fail. The number of traces follows the information that the two shares carry, $T \approx c H/(nI)$, so a profiled estimate of this information predicts the cost of the attack. A primal lattice step does not reduce the number of traces, because below the recovery threshold the marginals are calibrated but too diffuse; this conclusion rests on block-size estimates with diagonal hints, not on BKZ runs. Our evaluation covers one device and an implementation that loads individual arithmetic coefficient shares; bitsliced conversions, higher-order masking, and shuffling remain untested.

References

- [ADPS16] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum Key Exchange—A New Hope. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 327–343, Austin, TX, August 2016. USENIX Association.
- [AGVW17] Martin R. Albrecht, Florian Göpfert, Fernando Virdia, and Thomas Wunderer. Revisiting the Expected Cost of Solving uSVP and Applications to LWE. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology—ASIACRYPT 2017*, volume 10624 of *Lecture Notes in Computer Science*, pages 297–322. Springer, 2017.
- [APS15] Martin R. Albrecht, Rachel Player, and Sam Scott. On the Concrete Hardness of Learning with Errors. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015.
- [BBB⁺24] Estuardo Alpirez Bock, Gustavo Banegas, Chris Brzuska, Lukasz Chmielewski, Kirthivaasan Puniamurthy, and Milan Sorf. Breaking DPA-Protected Kyber via the Pair-Pointwise Multiplication. In Christina Pöpper and Lejla Batina, editors, *Applied Cryptography and Network Security - 22nd International Conference, ACNS 2024, Abu Dhabi, United Arab Emirates, March 5-8, 2024, Proceedings, Part II*, volume 14584 of *Lecture Notes in Computer Science*, pages 101–130. Springer, 2024.
- [BC22] Olivier Bronchain and Gaëtan Cassiers. Bitslicing Arithmetic/Boolean Masking Conversions for Fun and Profit with Application to Lattice-Based KEMs. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(4), 2022.
- [BDGL16] Anja Becker, Léo Ducas, Nicolas Gama, and Thijs Laarhoven. New Directions in Nearest Neighbor Searching with Applications to Lattice Sieving. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2016)*, pages 10–24. Society for Industrial and Applied Mathematics, 2016.
- [BDK⁺21] Joppe Bos, Léo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Kyber: Algorithm specifications and supporting documentation, version 3.01, January 2021. Round-3 submission to the NIST Post-Quantum Cryptography Standardization Project.
- [BDV21] Michiel Van Beirendonck, Jan-Pieter D’Anvers, and Ingrid Verbauwhede. Analysis and Comparison of Table-based Arithmetic to Boolean Masking. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(3):275–297, 2021.
- [BG14] Shi Bai and Steven D. Galbraith. Lattice Decoding Attacks on Binary LWE. In Willy Susilo and Yi Mu, editors, *Information Security and Privacy—ACISP 2014*, volume 8544 of *Lecture Notes in Computer Science*, pages 322–337. Springer, 2014.
- [BGR⁺21] Joppe W. Bos, Marc Gourjon, Joost Renes, Tobias Schneider, and Christine van Vredendaal. Masking Kyber: First- and Higher-Order Implementations. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(4):173–214, 2021.
- [BHM⁺19] Olivier Bronchain, Julien M. Hendrickx, Clément Massart, Alex Olshevsky, and François-Xavier Standaert. Leakage Certification Revisited: Bounding

- Model Errors in Side-Channel Security Evaluations. In *Advances in Cryptology – CRYPTO 2019*, Lecture Notes in Computer Science. Springer, 2019.
- [BNGD23] Linus Backlund, Kalle Ngo, Joel Gärtner, and Elena Dubrova. Secret Key Recovery Attack on Masked and Shuffled Implementations of CRYSTALS-Kyber and Saber. In *Applied Cryptography and Network Security Workshops – ACNS 2023*, Lecture Notes in Computer Science, pages 159–177. Springer, 2023.
- [CMST25] Kévin Carrier, Charles Meyer-Hilfiger, Yixin Shen, and Jean-Pierre Tillich. Assessing the Impact of a Variant of MATZOV’s Dual Attack on Kyber. In Yael Tauman Kalai and Seny F. Kamara, editors, *Advances in Cryptology – CRYPTO 2025 - 45th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2025, Proceedings, Part I*, volume 16000 of *Lecture Notes in Computer Science*, pages 444–476. Springer, 2025.
- [CN11] Yuanmi Chen and Phong Q. Nguyen. BKZ 2.0: Better Lattice Security Estimates. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011 - 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings*, volume 7073 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2011.
- [CRR02] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template Attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*, pages 13–28. Springer, 2002.
- [DDGR20] Dana Dachman-Soled, Léo Ducas, Huijing Gong, and Mélissa Rossi. LWE with Side Information: Attacks and Concrete Security Estimation. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part II*, volume 12171 of *Lecture Notes in Computer Science*, pages 329–358. Springer, 2020.
- [Del22] Jeroen Delvaux. Roulette: A Diverse Family of Feasible Fault Attacks on Masked Kyber. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(4):637–660, 2022.
- [DGHK23] Dana Dachman-Soled, Huijing Gong, Tom Hanson, and Hunter Kippen. Revisiting Security Estimation for LWE with Hints from a Geometric Perspective. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part V*, volume 14085 of *Lecture Notes in Computer Science*, pages 748–781. Springer, 2023.
- [DMM09] David L. Donoho, Arian Maleki, and Andrea Montanari. Message-passing algorithms for compressed sensing. *Proceedings of the National Academy of Sciences*, 106(45):18914–18919, 2009.
- [DNGW23] Elena Dubrova, Kalle Ngo, Joel Gärtner, and Ruize Wang. Breaking a Fifth-Order Masked Implementation of CRYSTALS-Kyber by Copy-Paste. In *Proceedings of the 10th ACM Asia Public-Key Cryptography Workshop (APKC)*, pages 10–20. ACM, 2023.

- [GN08] Nicolas Gama and Phong Q. Nguyen. Predicting Lattice Reduction. In Nigel P. Smart, editor, *Advances in Cryptology—EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 31–51. Springer, 2008.
- [Hes02] Tom Heskes. Stable Fixed Points of Loopy Belief Propagation Are Local Minima of the Bethe Free Energy. In *Advances in Neural Information Processing Systems 15*, pages 343–350. MIT Press, 2002.
- [HHP⁺21] Mike Hamburg, Julius Hermelink, Robert Primas, Simona Samardjiska, Thomas Schamberger, Silvan Streit, Emanuele Strieder, and Christine van Vredendaal. Chosen Ciphertext k-Trace Attacks on Masked CCA2 Secure Kyber. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(4):88–113, 2021.
- [HKL⁺22] Daniel Heinz, Matthias J. Kannwischer, Georg Land, Thomas Pöppelmann, Peter Schwabe, and Amber Sprenkels. First-Order Masked Kyber on ARM Cortex-M4. Cryptology ePrint Archive, Paper 2022/058, 2022.
- [HMS⁺23] Julius Hermelink, Erik Mårtensson, Simona Samardjiska, Peter Pessl, and Gabi Dreier Rodosek. Belief Propagation Meets Lattice Reduction: Security Estimates for Error-Tolerant Key Recovery from Decryption Errors. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2023(4):287–317, 2023.
- [HPP21] Julius Hermelink, Peter Pessl, and Thomas Pöppelmann. Fault-Enabled Chosen-Ciphertext Attacks on Kyber. In Avishek Adhikari, Ralf Küsters, and Bart Preneel, editors, *Progress in Cryptology - INDOCRYPT 2021 - 22nd International Conference on Cryptology in India, Jaipur, India, December 12-15, 2021, Proceedings*, volume 13143 of *Lecture Notes in Computer Science*, pages 311–334. Springer, 2021.
- [HSMP25] Julius Hermelink, Silvan Streit, Erik Mårtensson, and Richard Petri. A Generic Framework for Side-Channel Attacks Against LWE-Based Cryptosystems. In Serge Fehr and Pierre-Alain Fouque, editors, *Advances in Cryptology - EUROCRYPT 2025 - 44th Annual International Conference on the Theory and Applications of Cryptographic Techniques Madrid, Spain, May 4-8, 2025, Proceedings, Part VIII*, volume 15608 of *Lecture Notes in Computer Science*, pages 3–32. Springer, 2025.
- [JD25] Yanning Ji and Elena Dubrova. A side-channel attack on a masked hardware implementation of CRYSTALS-Kyber. *J. Cryptogr. Eng.*, 15(1):7, 2025.
- [Kab03] Yoshiyuki Kabashima. A CDMA Multiuser Detection Algorithm on the Basis of Belief Propagation. *Journal of Physics A: Mathematical and General*, 36(43):11111–11121, 2003.
- [Kan87] Ravi Kannan. Minkowski’s Convex Body Theorem and Integer Programming. *Mathematics of Operations Research*, 12(3):415–440, 1987.
- [KT24] Yen-Ting Kuo and Atsushi Takayasu. A Lattice Attack on CRYSTALS-Kyber with Correlation Power Analysis. In Hwajeong Seo and Suhri Kim, editors, *Information Security and Cryptology – ICISC 2023*, pages 202–220, Singapore, 2024. Springer Nature Singapore.
- [LFP25] Qian Lu, Yansong Feng, and Yanbin Pan. Solving LWE with Independent Hints about Secret and Errors. Cryptology ePrint Archive, Paper 2025/1128, 2025.

- [LHWLL82] Arjen K. Lenstra, Jr. Hendrik W. Lenstra, and László Lovász. Factoring Polynomials with Rational Coefficients. *Mathematische Annalen*, 261(4):515–534, 1982.
- [MN23] Alexander May and Julian Nowakowski. Too Many Hints - When LLL Breaks LWE. In Jian Guo and Ron Steinfeld, editors, *Advances in Cryptology - ASIACRYPT 2023 - 29th International Conference on the Theory and Application of Cryptology and Information Security, Guangzhou, China, December 4-8, 2023, Proceedings, Part IV*, volume 14441 of *Lecture Notes in Computer Science*, pages 106–137. Springer, 2023.
- [Nat24] National Institute of Standards and Technology. Module-Lattice-Based Key-Encapsulation Mechanism Standard. Federal Information Processing Standards Publication NIST FIPS 203, U.S. Department of Commerce, August 2024.
- [NDGJ21] Kalle Ngo, Elena Dubrova, Qian Guo, and Thomas Johansson. A Side-Channel Attack on a Masked IND-CCA Secure Saber KEM Implementation. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(4):676–707, 2021.
- [OSPG18] Tobias Oder, Tobias Schneider, Thomas Pöppelmann, and Tim Güneysu. Practical CCA2-Secure and Masked Ring-LWE Implementation. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(1):142–174, 2018.
- [PP21] Peter Pessl and Lukas Prokop. Fault Attacks on CCA-secure Lattice KEMs. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(2):37–60, 2021.
- [PPM17] Robert Primas, Peter Pessl, and Stefan Mangard. Single-Trace Side-Channel Attacks on Masked Lattice-Based Encryption. In *Cryptographic Hardware and Embedded Systems – CHES 2017*, volume 10529 of *Lecture Notes in Computer Science*, pages 513–533. Springer, 2017.
- [Pre05] Marco Pretti. A Message-Passing Algorithm with Damping. *Journal of Statistical Mechanics: Theory and Experiment*, 2005(11):P11008, 2005.
- [PS24] Amaury Pouly and Yixin Shen. Provable Dual Attacks on Learning with Errors. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26-30, 2024, Proceedings, Part VI*, volume 14656 of *Lecture Notes in Computer Science*, pages 256–285. Springer, 2024.
- [QWC⁺24] Kexin Qiao, Zhaoyang Wang, Heng Chang, Siwei Sun, Zehan Wu, Junjie Cheng, Changhai Ou, An Wang, and Liehuang Zhu. A closer look at the belief propagation algorithm in side-channel attack on CCA-secure PQC KEM. *Sci. China Inf. Sci.*, 67(11), 2024.
- [Ran10] Sundeep Rangan. Estimation with Random Linear Mixing, Belief Propagation and Compressed Sensing. In *44th Annual Conference on Information Sciences and Systems (CISS)*. IEEE, 2010.
- [Ran11] Sundeep Rangan. Generalized Approximate Message Passing for Estimation with Random Linear Mixing. In *2011 IEEE International Symposium on Information Theory Proceedings*, pages 2168–2172. IEEE, 2011.

- [RPJ⁺24] Prasanna Ravi, Thales B. Paiva, Dirmanto Jap, Jan-Pieter D’Anvers, and Shivam Bhasin. Defeating Low-Cost Countermeasures against Side-Channel Attacks in Lattice-based Encryption A Case Study on Crystals-Kyber. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2024(2):795–818, 2024.
- [RRCB19] Prasanna Ravi, Sujoy Sinha Roy, Anupam Chattopadhyay, and Shivam Bhasin. Generic Side-channel attacks on CCA-secure lattice-based PKE and KEM schemes. *IACR Cryptol. ePrint Arch.*, 2019:948, 2019.
- [Sch03] Claus Peter Schnorr. Lattice Reduction by Random Sampling and Birthday Methods. In Helmut Alt and Michel Habib, editors, *STACS 2003*, volume 2607 of *Lecture Notes in Computer Science*, pages 145–156. Springer, 2003.
- [SE91] Claus-Peter Schnorr and Martin Euchner. Lattice Basis Reduction: Improved Practical Algorithms and Solving Subset Sum Problems. volume 529 of *Lecture Notes in Computer Science*, pages 68–85. Springer, 1991.
- [SL03] George A. F. Seber and Alan J. Lee. *Linear Regression Analysis*. John Wiley & Sons, 2 edition, 2003.
- [VGS14] Nicolas Veyrat-Charvillon, Benoît Gérard, and François-Xavier Standaert. Soft Analytical Side-Channel Attacks. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014. Proceedings, Part I*, volume 8873 of *Lecture Notes in Computer Science*, pages 282–296. Springer, 2014.
- [WBD24] Ruize Wang, Martin Brisfors, and Elena Dubrova. A Side-Channel Attack on a Higher-Order Masked CRYSTALS-Kyber Implementation. In Christina Pöpper and Lejla Batina, editors, *Applied Cryptography and Network Security*, pages 301–324, Cham, 2024. Springer Nature Switzerland.
- [WXT25] Kai Wang, Dejun Xu, and Jing Tian. An Improved Two-Step Attack on Lattice-Based Cryptography: A Case Study of Kyber. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 44(9):3643–3647, 2025.
- [XPRO20] Zhuang Xu, Owen Pemberton, Sujoy Sinha Roy, and David F. Oswald. Magnifying Side-Channel Leakage of Lattice-Based Cryptosystems with Chosen Ciphertexts: The Case Study of Kyber. *IACR Cryptol. ePrint Arch.*, 2020:912, 2020.