Recovering HQC Secret Key from Distance Spectrum using Side-Channel Attack

Magali Salom[1,2], Angélique Lopez[2], Valentin Vasseur[2] and Gabriel Zaid[3]

[1] INRIA, Paris, France, {firstname.lastname}@inria.fr
[2] Thales, Gennevilliers, France, {firstname.lastname}@thalesgroup.com
[3] CryptoExperts, Paris, France, {firstname.lastname}@cryptoexperts.com

Abstract. HQC is a code-based key encapsulation mechanism selected for standardization by NIST. It uses sparse secret vectors in a quasi-cyclic structure, where GJS-style attacks can exploit correlations between private syndrome weight and the secret-key distance spectrum. However, HQC decapsulation does not directly manipulate any syndrome, leaving open whether any such attack could be performed in practice.
In this work, we present the first practical end-to-end side-channel attack against HQC that recovers the secret key through error-weight leakage and distance-spectrum reconstruction. We identify two sources of leakage in the Reed–Muller decoder under different attacker capabilities. In a chosen-ciphertext attack (CCA), we exploit the ExpandAndSum function by imposing a zero message, whereas in a known-ciphertext attack (KCA), we target the FindPeaks function without modifying the legitimate encapsulation procedure. We provide a theoretical characterization of the Hadamard coefficients manipulated by FindPeaks, establishing a direct relationship between the highest peak and the error weight under a sufficient decoding condition, while showing that the remaining coefficients also depend on the error pattern.
We develop profiled side-channel attacks to estimate the error weight and construct a soft distance spectrum of the secret key. To recover the key from imperfect information, we introduce a simulated-annealing reconstruction algorithm that, in simulations, reduces the number of required error-weight observations by approximately 5.8× compared with the state-of-the-art SSV26 method, with substantially lower reconstruction times. Our experiments on an ARM Cortex-M4 implementation of HQC-1 demonstrate end-to-end secret-key recovery in both attack settings. In the CCA setting, our algorithm achieves a 93.5% success rate with 40 000 decapsulation traces, compared to 200 000 traces for SSV26, and recovers the secret key in all 200 trials with 100 000 traces. Most notably, in the more restrictive KCA setting, our approach recovers the secret key in all 200 trials using 1 000 000 traces, whereas the reconstruction algorithm from SSV26 fails to recover the key in any of the 200 trials, even with 1 300 000 traces.

Keywords: Post-quantum cryptography · HQC · Side-channel attacks · Reed–Muller decoding · Error-weight leakage · Secret-key recovery

1 Introduction

Context. The development of quantum computers has motivated the transition toward post-quantum cryptographic algorithms. In this context, the National Institute of Standards and Technology (NIST) has conducted a standardization process to select cryptographic schemes resistant to attacks by quantum computers. Among these schemes, Hamming Quasi-Cyclic (HQC) was selected for standardization in 2025 as a code-based alternative to the lattice-based key encapsulation mechanism ML-KEM.

HQC is a code-based key encapsulation mechanism (KEM) whose security relies on the hardness of the quasi-cyclic syndrome decoding problem [GAMA+25]. During decapsulation, the recipient performs error correction using a concatenated code composed of Reed–Solomon (RS) and duplicated Reed–Muller (RM) codes. Although the security of HQC is based on a computationally hard problem, its implementation may reveal sensitive information through physical side channels, such as power consumption, electromagnetic emanations, or execution timing.

Several side-channel attacks against HQC have demonstrated the sensitivity of its error-correction procedure. In particular, previous works have exploited leakage from the RM decoder and its different components to recover secret information [GLG22, PRJ+25, MNMD25, BMG+25]. More recently, the FindPeaks function has also been targeted through cache-timing attacks [DG26]. These results highlight the importance of investigating the intermediate computations performed during HQC decapsulation.

An alternative approach to secret-key recovery consists in recovering the distance spectrum of a secret sparse vector from key-dependent decoding behavior and then reconstructing the vector from this spectrum. The spectrum describes the cyclic distances between the nonzero coefficients of a secret polynomial, counted with multiplicity. This approach originates in the reaction attack of Guo, Johansson, and Stankovski (GJS) against a QC-MDPC McEliece cryptosystem, which exploits decoding failures [GJS16]. [ELPS18] related this attack to correlations with the syndrome-weight distribution from which they derived a timing attack. For an earlier BCH-based version of HQC, [PT19] combined decoder timing with randomized reconstruction from partial spectrum information.

Several algorithms have been proposed for reconstruction from hard spectrum information [FHS+17, PT18, OIT+25]. For side-channel attacks, however, algorithms that can use imperfect information are particularly relevant. [OII+25] adapted depth-first and breadth-first search to imperfect spectra represented by lower and upper bounds on the set of occurring distances. More recently, [SSV26] proposed reconstruction from a soft distance spectrum, represented by a probability distribution over the multiplicity of each distance.

The latter work primarily targets BIKE, a QC-MDPC-based KEM, under an idealized model in which the adversary can directly observe the syndrome weight. Its applicability to HQC, however, remained hypothetical. Indeed with BIKE, the decoder actually computes and performs operations on the syndrome, but for HQC, the relevant quantity is an error vector to a random codeword, and this error is never directly manipulated by the decoder.

Exploiting this approach in practice requires recovering sufficiently accurate error-weight information from physical leakage. Moreover, the recovered observations may be noisy, making the reconstruction of the secret key more challenging. This raises two questions: how can the Hamming weight of the decryption error be estimated from HQC side-channel leakage, and how can these estimates be efficiently exploited for secret-key recovery?

Contributions. In this work, we investigate the recovery of HQC secret keys from side-channel leakage associated with the Hamming weight of the decryption error. Our contributions are summarized as follows.

- **Identification of error-weight leakage in the RM decoder.** We investigate two functions of the HQC Reed–Muller decoder that manipulate intermediate values related to the Hamming weight of the decryption error. We first consider a chosen-ciphertext attack (CCA) targeting the ExpandAndSum function, where the attacker can impose a zero message. We then consider a known-ciphertext attack (KCA) targeting the FindPeaks function, where encapsulation follows the legitimate protocol. Both settings enable the attacker to obtain information about the error weight through side-channel leakage.

- **Theoretical characterization of the Hadamard coefficients.** We establish a direct relationship between the amplitude of the correct Hadamard coefficient and the Hamming weight of the corresponding RM error block. In particular, we show that, under a sufficient condition on the number of errors, the highest Hadamard peak directly determines this weight. We also characterize the remaining Hadamard coefficients, showing that their values depend on both the Hamming weight and the positions of the errors. These results provide a theoretical basis for investigating error-weight leakage throughout the execution of FindPeaks.

- **Improved secret-key reconstruction from imperfect information.** Building upon the soft distance spectrum approach introduced in [SSV26], we propose a new key-reconstruction algorithm based on simulated annealing. Our algorithm explores candidate keys using transformations that preserve their Hamming weight and a score derived from the soft spectrum. Compared to the previous reconstruction algorithm, our approach significantly reduces the number of error-weight observations required for successful key recovery. In particular, our simulations show a reduction by a factor of 4.4 to achieve a success rate of 50%, and approximately 5.8 to achieve a success rate of 95%. These improvements are maintained in the presence of noisy error-weight observations.

- **Experimental validation and end-to-end key recovery.** We evaluate both side-channel attacks on an ARM Cortex-M4 implementation of HQC. For HQC-1, our experiments achieve an empirical success rate of 100% over 200 trials with 100 000 decapsulation traces in the CCA setting and 1 000 000 traces in the KCA setting. Moreover, in the CCA setting, our approach achieves a success rate of 93.5% with 40 000 traces, whereas the reconstruction algorithm from [SSV26] requires 200 000 traces to reach the same success rate, corresponding to a fivefold reduction. In the KCA setting, the previous reconstruction algorithm did not recover the correct secret key in any of the 200 trials performed with 1 300 000 traces.

- **Countermeasures and additional leakage sources.** We investigate the effectiveness of masking against the identified error-weight leakage, considering a masked implementation of the RM decoder [GSA+25]. We also investigate the Reed–Solomon (RS) decoder as an additional potential source of error-weight leakage. Using the probabilistic model from [BDM+26, Model 2], we characterize the relationship between the number of erroneous RS symbols and the Hamming weight of the full error pattern. Our analysis shows that this quantity provides statistical information about the error weight, although the resulting observations are considerably coarser than those obtained from the RM decoder. These findings motivate further investigations into potential leakage from RS decoding and the protection of HQC's complete error-correction procedure.

Overall, these investigations provide practical evidence that the Hamming weight of the decryption error can be exposed through side-channel leakage and exploited for secret-key recovery. Our results highlight the importance of protecting intermediate computations performed during HQC error correction, even when they do not directly manipulate secret-key coefficients. Furthermore, our analysis of masking countermeasures and potential leakage from Reed–Solomon decoding emphasizes the need to consider the complete error-correction procedure when evaluating the side-channel security of HQC implementations.

Paper organization. The remainder of this paper is organized as follows. Section 2 introduces HQC, the Reed–Muller decoding procedure, and existing side-channel attacks against HQC. Section 3 identifies the sources of error-weight leakage and presents two

new attack surfaces considering both chosen-ciphertext and known-ciphertext attacks. Section 4 describes the recovery of the secret key from a soft distance spectrum and introduces our reconstruction algorithm based on simulated annealing. Section 5 presents the experimental setup and evaluates the end-to-end attacks on real device. Section 6 discusses potential countermeasures and further attack opportunities, before Section 7 concludes the paper.

2 Preliminaries

2.1 Notations

We denote by $\mathbb{F}_q$ the finite field with q elements, and by $\mathbb{Z}$ the set of integers. Vectors are denoted by lowercase letters (*e.g.*, $\mathbf{x}$) and matrices by uppercase letters (*e.g.*, $\mathbf{X}$). For a binary vector $\mathbf{x}$, we denote its Hamming weight by $|\mathbf{x}|$ and its support by $\mathrm{supp}(\mathbf{x}) = \{i \mid x_i = 1\}$. For a finite set $\mathcal{U}$, $|\mathcal{U}|$ denotes its cardinality. The notation $u \xleftarrow{\$} \mathcal{U}$ indicates that u is sampled uniformly at random from a finite set $\mathcal{U}$. We denote by $\Pr[\cdot]$, $\mathbb{E}[\cdot]$, and $\mathrm{Var}[\cdot]$ the probability, expectation, and variance operators, respectively. The Gaussian distribution with mean μ and variance σ^2 is denoted by $\mathcal{N}(\mu, \sigma^2)$. Finally, we denote by $\mathcal{R} = \mathbb{F}_2[X]/(X^n - 1)$ the ring of binary polynomials modulo $X^n - 1$. Elements of $\mathcal{R}$ are identified with binary vectors of length n. For a non-negative integer ω, we define $\mathcal{R}_\omega = \{\mathbf{x} \in \mathcal{R} \mid |\mathbf{x}| = \omega\}$.

2.2 Hamming Quasi-Cyclic algorithm

HQC [GAMA+25] is a code-based Key Encapsulation Mechanism (KEM) whose security relies on the hardness of the quasi-cyclic syndrome decoding problem. HQC is constructed from an underlying Public Key Encryption (PKE) scheme, detailed in [GAMA+25, Section 3.5]. The KEM is obtained by applying the HHK transform [HHK17], which converts the IND-CPA-secure PKE into an IND-CCA-secure KEM. Algorithms 1, 2, and 3 describe the resulting HQC KEM. In this work, we focus on the PKE decryption performed during decapsulation, as highlighted in Algorithm 3.

Algorithm 1 HQC-KEM.KeyGen

Ensure: sk, pk

1: $(\mathbf{x}, \mathbf{y}) \xleftarrow{\$} \mathcal{R}_\omega^2$
2: $\mathbf{h} \xleftarrow{\$} \mathcal{R}$
3: $\mathbf{s} \leftarrow \mathbf{x} + \mathbf{h}\mathbf{y}$
4: **return** $\mathrm{pk} = (\mathbf{h}, \mathbf{s}), \mathrm{sk} = (\mathbf{x}, \mathbf{y})$

Algorithm 2 HQC-KEM.Encaps

Require: pk

Ensure: $K, (\mathbf{c}, \mathrm{salt})$

1: $\mathbf{m} \xleftarrow{\$} \mathbb{F}_2^k$; $\mathrm{salt} \xleftarrow{\$} \mathbb{F}_2^{128}$
2: $(K, \theta) \leftarrow \mathcal{G}(\mathcal{H}(\mathrm{pk}) \| \mathbf{m} \| \mathrm{salt})$
3: $\mathbf{c} \leftarrow \text{HQC-PKE.Encrypt}(\mathrm{pk}, \mathbf{m}, \theta)$
4: **return** $K, \mathbf{c}, \mathrm{salt}$

Algorithm 3 HQC-KEM.Decaps

Require: sk, pk, $\mathbf{c}$, salt, σ

Ensure: K'

1: $\mathbf{m}' \leftarrow$ HQC-PKE.Decrypt(sk, $\mathbf{c}$) ▷ Function analyzed in this paper.
2: $(K', \theta') \leftarrow \mathcal{G}(\mathcal{H}(\mathrm{pk}) \| \mathbf{m}' \| \mathrm{salt})$
3: $\mathbf{c}' \leftarrow \text{HQC-PKE.Encrypt}(\mathrm{pk}, \mathbf{m}', \theta')$
4: **if** $\mathbf{m}' = \perp$ or $\mathbf{c} \neq \mathbf{c}'$ **then**
5: $\quad K' \leftarrow \mathcal{J}(\mathcal{H}(\mathrm{pk}) \| \sigma \| \mathbf{c})$
6: **return** K'

During PKE decryption, the error vector is given by $\mathbf{e}' = \text{Truncate}(\mathbf{xr_2} - \mathbf{yr_1} + \mathbf{e}, n_1 n_2)$ so that the receiver computes a noisy codeword of the form $\mathcal{C}.\text{Encode}(\mathbf{m}) + \mathbf{e}'$. Here, $\mathbf{x}$ and $\mathbf{y}$ are the sparse secret-key polynomials, while $\mathbf{r_1}$, $\mathbf{r_2}$, and $\mathbf{e}$ are the sparse ephemeral polynomials sampled during PKE encryption. The code $\mathcal{C}$ is then used to recover $\mathbf{m}$ from this noisy codeword. HQC uses a public code $\mathcal{C}$ constructed by concatenating two error-correcting codes. A concatenated code is formed from an outer code $[n_1, k_1, d_1]$ over $\mathbb{F}_q$ and an inner code $[n_2, k_2, d_2]$ over $\mathbb{F}_2$, with $q = 2^{k_2}$. In HQC, the outer code is a shortened Reed–Solomon (RS) code, while the inner code is a duplicated Reed–Muller (RM) code.

A message $\mathbf{m} \in \mathbb{F}_2^k$, with $k = k_1 k_2$, is first interpreted as k_1 symbols over $\mathbb{F}_q$ and encoded by the RS encoder into an RS codeword of length n_1 over $\mathbb{F}_q$. Then, an RM encoder is applied independently to each of the n_1 symbols, resulting in a binary codeword of length $n_1 n_2$. Decoding proceeds in the reverse order. The noisy codeword is split into n_1 blocks of length n_2, and each block is decoded independently using an RM decoder. The resulting sequence of n_1 symbols is then decoded using an RS decoder. The HQC parameter sets considered in [GAMA+25] are summarized in Table 1.

Table 1: HQC parameters from [GAMA+25].

Security	mul	k_1	k_2	n_1	n_2	$n_1 n_2$	n	l	ω	$\omega_e = \omega_r$
HQC-1	3	16	8	46	384	17664	17669	5	66	75
HQC-3	5	24	8	56	640	35840	35851	11	100	114
HQC-5	5	32	8	90	640	57600	57637	37	131	149

The Hamming weight of the error $\mathbf{e}'$ has been identified in [SSV26] as a quantity that can be exploited for key recovery. Indeed, knowledge of multiple error weights can lead to the recovery of the distance spectrum of the secret key $\mathbf{y}$, which describes the cyclic distances between its nonzero coefficients, counted with multiplicity. A reconstruction algorithm can then recover the secret key from its spectrum. Our attack therefore aims at recovering $|\mathbf{e}'|$ by exploiting side-channel leakage from the RM decoding process, which we present next.

2.3 Reed–Muller Decoding in HQC

HQC uses a first-order Reed–Muller code RM(1,7) with parameters $[128, 8, 64]$. This RM code is duplicated a number of times denoted by *mul*, which is set to 3 for HQC-1 and 5 for HQC-3 and HQC-5. This results in a code with parameters $[384, 8, 192]$ and $[640, 8, 320]$, respectively, which is decoded using a Fast Hadamard Transform (FHT). As explained in the previous section, the decoding process splits the $n_1 n_2$-bit noisy codeword $\mathbf{mG} + \mathbf{e}'$ into n_1 blocks of n_2 bits, and applies an RM decoder to each block independently. Each RM decoder aims to recover one symbol of the Reed–Solomon codeword from its corresponding noisy block, so that the subsequent RS decoder can correct any remaining symbol errors and recover the original message. In particular, for HQC-1, the decoding process consists of 46 independent RM decodings, each operating on a 384-bit block. Each RM decoding consists of the following three steps:

- The ExpandAndSum function takes as input *mul* noisy codewords of 128 bits and combines them into a single vector of 128 integers by summing the corresponding bits across the copies.
- The Fast Hadamard Transform is then applied to the output of ExpandAndSum to compute the coefficients used to identify the most likely transmitted RM codeword.
- The FindPeaks function finds the coefficient with the largest absolute value in the FHT output. Its position and sign are used to recover the decoded symbol.

The complete RM decoding procedure is detailed in Algorithm 4.

Algorithm 4 Reed–Muller decoding.

Require: $\mathbf{c} = \mathbf{mG} + \mathbf{e}' = (c_0, \ldots, c_{n_1 n_2 - 1}) \in \mathbb{F}_2^{n_1 n_2}$
Ensure: $\mathbf{m}' = (m'_0, \ldots, m'_{n_1 - 1}) \in \mathbb{F}_{2^8}^{n_1}$
1: **for** $i = 0$ to $n_1 - 1$ **do**
2: $\quad \mathbf{u} \leftarrow \texttt{ExpandAndSum}(\mathbf{c}[i \cdot n_2, \ldots, (i+1) \cdot n_2 - 1])$
3: $\quad \mathbf{H} \leftarrow \texttt{Hadamard}(\mathbf{u})$
4: $\quad H_0 \leftarrow H_0 - 64 \times mul$
5: $\quad m'_i \leftarrow \texttt{FindPeaks}(\mathbf{H})$
6: **return** $\mathbf{m}'$

Throughout the RM decoding process, the error $\mathbf{e}'$ is never manipulated independently, but always as part of the noisy codeword $\mathbf{mG} + \mathbf{e}'$. This makes the extraction of information about the error through side-channel leakage more challenging. Nevertheless, the intermediate computations performed during RM decoding may reveal information about the Hamming weight of $\mathbf{e}'$, as investigated in this work. We next review existing side-channel attacks on HQC before investigating this leakage in detail.

2.4 Existing Side-Channel Attacks on HQC

Side-channel attacks (SCAs) are a class of cryptographic attacks in which an adversary exploits implementation-level vulnerabilities of cryptographic systems by analyzing information leaked during their execution. During cryptographic operations, side-channel traces correlated with key-dependent intermediate variables can be captured. These traces can then be analyzed using statistical methods to distinguish between key hypotheses and recover secret information.

HQC has been the target of several side-channel attacks exploiting timing, cache behavior, power consumption, or electromagnetic (EM) emanations. Most existing attacks focus on the decapsulation procedure and, in particular, on the error-correction mechanisms used during decryption. Previous works have targeted different versions of the HQC reference implementation, showing that leakage arising from the decoding process can be exploited to recover secret information. In particular, several works have focused on Reed–Muller (RM) decoding which is also targeted in this paper. In [GLG22], the authors exploit leakage from the Fast Hadamard Transform (FHT) to construct an error-weight oracle, which they use with a divide-and-conquer strategy in a chosen-ciphertext attack scenario. The FHT was also targeted by a soft analytical side-channel attack in [BMG+25] to recover the shared key from a single attack trace. The authors also propose two chosen-ciphertext attacks to recover the secret key by exploiting the same FHT leakage. Other works have targeted different components of the RM decoder. In particular, the ExpandAndSum function was attacked in [MNMD25, PRJ+25, DGN26, GHL+26], while the FindPeaks function was exploited in [PRJ+25, DG26] through a cache-timing attack to recover the secret key. Furthermore, [DG25a, DG25b] propose oracle-based side-channel attacks on HQC that exploit the outcome of RM decoding to recover the secret key positions.

In this work, we focus on *profiled side-channel attacks*. In this setting, the adversary is assumed to have access to a profiling device that is identical to the target device and over which they have full control. During the profiling phase, labeled traces corresponding to known sensitive intermediate values are collected and used to characterize their leakage. The resulting model is then applied to traces acquired from the target device to infer the corresponding sensitive values. Classical template attacks [CRR03], for example, model each leakage class using a multivariate Gaussian distribution characterized by its mean

vector and covariance matrix, and rank candidate values according to their likelihood. Such a profiling framework is particularly relevant to our attack on HQC, as it enables the leakage associated with specific intermediate values manipulated during decapsulation to be characterized and subsequently exploited for key recovery.

In the following section, we extend the current literature by investigating how leakage from the ExpandAndSum and FindPeaks functions can be exploited to estimate the Hamming weight of the error $\mathbf{e}'$, which is subsequently used for secret-key recovery.

3 Recovering the Error Weight

In this work, we investigate two sources of side-channel leakage that can be exploited by an adversary to retrieve information about the Hamming weight of the error $\mathbf{e}' = \text{Truncate}(\mathbf{x}\mathbf{r_2} - \mathbf{y}\mathbf{r_1} + \mathbf{e}, n_1 n_2)$. These weaknesses can be exploited under different adversarial capabilities. We therefore consider two attack scenarios: a chosen-ciphertext attack (CCA) and a known-ciphertext attack (KCA).

3.1 Chosen Ciphertext Attack

Threat model. In this scenario, we assume that the attacker knows the sparse vectors $\mathbf{r_1}, \mathbf{r_2}, \mathbf{e} \in \mathbb{F}_2^n$. This is a reasonable assumption since these vectors are sampled during encryption, which is performed by the attacker in our attack scenario. We also assume that the adversary can construct ciphertexts corresponding to a chosen message $\mathbf{m} \in \mathbb{F}_2^k$. This can be achieved by a small modification on HQC-KEM.Encaps which allows the adversary to specify $\mathbf{m}$ in input. To overcome the difficulty arising from the error being manipulated only as part of the noisy codeword, we focus on the case where $\mathbf{m} = \mathbf{0}$. In this setting, the encoded message is also zero, so the input to the decoding procedure reduces to $\mathbf{c} = \mathbf{m}\mathbf{G} + \mathbf{e}' = \mathbf{e}'$. Consequently, the intermediate computations performed during decoding directly depend on the error vector.

Weakness identification. As explained in Section 2.3, the decoding procedure starts by applying an RM decoder independently to each of the n_1 blocks of the noisy codeword. This process may present weaknesses that can be exploited to recover the Hamming weight of the error $\mathbf{e}'$.

The RM decoder takes as input a noisy codeword $\mathbf{c} = \mathbf{m}\mathbf{G} + \mathbf{e}'$ and outputs the corresponding decoded symbol $\mathbf{m}'$. It is composed of three functions: ExpandAndSum, Hadamard, and FindPeaks. Each function processes the output of the previous one, progressively transforming the noisy codeword. As a result, the relationship between the intermediate values and the original error becomes less direct.

The ExpandAndSum function is therefore a natural target, as it is the first step of the decoder and directly processes the noisy codeword. In particular, in our chosen-ciphertext setting where $\mathbf{m} = \mathbf{0}$, the noisy codeword reduces to $\mathbf{c} = \mathbf{e}'$. Consequently, the intermediate values manipulated during RM decoding depend directly on the error, potentially revealing information about its Hamming weight through side-channel leakage.

The ExpandAndSum function converts the received Reed–Muller noisy codeword into an expanded integer representation. The process is detailed in Algorithm 5. The function takes as input *mul* noisy codewords of 128 bits and outputs a vector where each element is obtained by summing the corresponding bits across the input blocks. More precisely, for the i-th RM block, with $0 \leq i < n_1$, the j-th element of the ExpandAndSum output vector $\mathbf{u} \in \mathbb{Z}^{128}$ is given by

$$u_j = \sum_{t=0}^{mul-1} c_{i \cdot n_2 + 128t + j}, \qquad 0 \leq j < 128, \tag{1}$$

where $n_2 = 128 \times mul$. Each coefficient u_j takes an integer value between 0 and mul. Since $\mathbf{c} = \mathbf{e}'$ in our attack setting, each coefficient u_j corresponds to the sum of mul error bits. Consequently, the Hamming weight of the error within the i-th RM block is given by

$$|\mathbf{e}'[i \cdot n_2, \ldots, (i+1) \cdot n_2 - 1]| = \sum_{k=i \cdot n_2}^{(i+1) \cdot n_2 - 1} e'_k = \sum_{j=0}^{127} u_j. \tag{2}$$

Thus, the output of ExpandAndSum directly determines the Hamming weight of the corresponding error block. Although this weight is not explicitly computed by the implementation, the manipulation of individual error bits and their partial sums may produce exploitable side-channel leakage.

Algorithm 5 Expand and Sum

Require: Noisy Reed–Muller block $\mathbf{c}[i \cdot n_2, \ldots, (i+1) \cdot n_2 - 1] \in \mathbb{F}_2^{n_2}$
Ensure: Expanded vector $\mathbf{u} \in \mathbb{Z}^{128}$

```
for j = 0 to 127 do
    u_j ← 0
for t = 0 to mul − 1 do
    for j = 0 to 127 do
        u_j ← u_j + c_{i·n2+128t+j}
return u
```

The resulting vector $\mathbf{u}$ is then used as input to the Hadamard function. For RM decoding, the Hadamard transform is naturally applied to a vector whose j-th element is $\sum_{t=0}^{mul-1} (-1)^{c_{i \cdot n_2 + 128t + j}}$. Since $(-1)^x = 1 - 2x$ for $x \in \mathbb{F}_2$, this expression can be rewritten as $\sum_{t=0}^{mul-1} (-1)^{c_{i \cdot n_2 + 128t + j}} = mul - 2u_j$. Due to the linearity of the Hadamard transform, we can directly apply the function to $\mathbf{u}$ and correct the resulting vector $\mathbf{H}$ afterwards, as shown in Algorithm 4. The Hadamard function also manipulates intermediate values related to the Hamming weight of the error. In particular, the first coefficient of the Hadamard transform, before correction, is the sum of the elements of $\mathbf{u}$. After applying the correction described in Algorithm 4, we obtain

$$H_0 = \sum_{j=0}^{127} u_j - 64 \times mul = \sum_{k=i \cdot n_2}^{(i+1) \cdot n_2 - 1} e'_k - \frac{n_2}{2}.$$

Therefore, the Hadamard transform also processes values directly related to the Hamming weight of the error, making it another potential target for side-channel attacks.

Both ExpandAndSum and Hadamard therefore manipulate intermediate values related to the Hamming weight of the error, making them potential targets for side-channel attacks. In particular, the first coefficient of the Hadamard transform is directly related to the error weight. However, the Hadamard transform involves several stages of additions and subtractions, making it potentially more challenging to isolate the leakage associated with this coefficient among the numerous intermediate computations. In contrast, ExpandAndSum directly manipulates the error bits and their partial sums, providing a more straightforward target for leakage analysis. We therefore focus on this function in our chosen-ciphertext attack. We next analyze its implementation to identify operations that may reveal information about the Hamming weight of the error.

Implementation analysis. Figure 1 shows the C code of the targeted implementation. For each RM decoding, the input block $\mathbf{c}[i \cdot n_2, \ldots, (i+1) \cdot n_2 - 1]$ consists of mul copies

of 128 bits, each stored as four 32-bit words in the input array `src` (highlighted in red). The algorithm first extracts every individual bit of the first copy using bitwise operations (Lines 2 to 4). The extracted bits are then summed with the corresponding bits of the remaining copies (Lines 7 to 10). The number of copies is represented by the `MULTIPLICITY` variable (*i.e. mul*), which is set to 3 for HQC-1. The ExpandAndSum function directly manipulates the noisy codeword $\mathbf{c} = \mathbf{mG} + \mathbf{e}'$ and, in our chosen-ciphertext setting where $\mathbf{m} = \mathbf{0}$, it directly manipulates $\mathbf{e}'$. As established in Equation 2, the output vector $\mathbf{u}$ is directly related to the Hamming weight of the corresponding error block. Moreover, the function mainly consists of bit extractions, assignments, and additions, providing multiple opportunities for data-dependent leakage.

As shown in the code, the error is first manipulated in a `for` loop, during the initialization of the output array `dest`. The input is processed as four words of 32 bits. We can therefore expect leakage associated with the four iterations of the outer loop, potentially resulting in four similar patterns in the side-channel trace. The remaining copies are then summed with the first one. For HQC-1, this involves two additional copies, each consisting of four 32-bit words. We can therefore expect eight additional groups of error-dependent operations. Overall, the processing of the three copies involves twelve groups of 32-bit word operations, which may result in repeated leakage patterns throughout the execution of ExpandAndSum. This analysis is confirmed through leakage analysis proposed in Appendix A. Such leakage could be exploited to estimate the Hamming weight of each RM error block. Since HQC-1 performs 46 RM decoding operations per decapsulation, these estimates can subsequently be combined to recover information about the Hamming weight of the complete error vector $\mathbf{e}'$.

```
1  void expand_and_sum(rm_expanded_cdw *dest, rm_codeword_t src[]) {
2    for (int32_t part = 0; part < 4; part++) {
3      for (int32_t bit = 0; bit < 32; bit++) {
4        (*dest)[part*32 + bit] = src[0].u32[part] >> bit & 1;
5      }
6    }
7    for (int32_t copy = 1; copy < MULTIPLICITY; copy++) {
8      for (int32_t part = 0; part < 4; part++) {
9        for (int32_t bit = 0; bit < 32; bit++) {
10         (*dest)[part*32 + bit] += src[copy].u32[part] >> bit & 1;
11       }
12     }
13   }
14 }
```

Figure 1: C code of ExpandAndSum extracted from `reed_muller.c` [HQC]. The targeted variable is highlighted in red.

The chosen-ciphertext setting allows the attacker to force $\mathbf{m} = \mathbf{0}$, making the error $\mathbf{e}'$ directly accessible to the ExpandAndSum function. However, this assumption does not hold in a more restrictive setting where the attacker only observes legitimate ciphertexts. In this case, the decoder processes the noisy codeword $\mathbf{c} = \mathbf{mG} + \mathbf{e}'$, making it more challenging to isolate information about the error weight from the leakage of ExpandAndSum. We therefore investigate another potential source of leakage within the RM decoder.

3.2 Known Ciphertext Attack

Threat model. We now consider a known-ciphertext attack (KCA) scenario where the attacker knows the sparse vectors $\mathbf{r_1}, \mathbf{r_2}, \mathbf{e} \in \mathbb{F}_2^n$ and the ciphertext $\mathbf{c} \in \mathbb{F}_2^{n_1 n_2}$. Unlike the chosen-ciphertext setting, the attacker does not modify the encapsulation procedure to impose a specific message. Instead, ciphertexts are generated according to the legitimate protocol. Consequently, the RM decoder processes the noisy codeword $\mathbf{c} = \mathbf{mG} + \mathbf{e}'$, rather than the error $\mathbf{e}'$ alone.

Weakness identification. We can still exploit leakage from the ExpandAndSum and Hadamard functions. However, these functions have already been targeted in the literature [GLG22, PRJ+25, MNMD25, BMG+25]. In this work, we choose to focus on a more subtle source of leakage involving an intermediate variable in the FindPeaks function that is directly related to the Hamming weight of the error. To the best of our knowledge, the exploitation of this intermediate value for error-weight estimation has not previously been investigated.

The FindPeaks algorithm (Algorithm 6) iterates over the 128 coefficients produced by the Fast Hadamard Transform. At each iteration, the absolute value of the current coefficient is compared against the current maximum. If it is larger, the current maximum is updated and the corresponding index is stored. The function returns the position of the coefficient with the largest absolute value, together with its sign, encoded as an 8-bit symbol corresponding to the RM information word.

Algorithm 6 Find Peaks

Require: $\mathbf{H} \in \mathbb{Z}^{128}$
Ensure: Decoded symbol $p \in \{0, \ldots, 255\}$
1: $v_{\max} \leftarrow 0$; $p \leftarrow 0$; $s \leftarrow 0$
2: **for** $j = 0$ to 127 **do**
3: **if** $abs(H_j) > v_{\max}$ **then**
4: $(v_{\max}, p, s) \leftarrow (abs(H_j), j, H_j)$
5: **if** $s > 0$ **then**
6: $p \leftarrow p + 128$
7: **return** p

An important observation is that the amplitude of the highest Hadamard peak is directly related to the Hamming weight of the error, even when the decoded message is nonzero. Indeed, the Hadamard transform exploits the structure of the RM code to identify the most likely transmitted codeword. When the correct peak is selected, its amplitude depends on the number of errors affecting the corresponding RM block. The following proposition establishes this relationship.

Proposition 1. *Let $w_i = |\mathbf{e}'[i \cdot n_2, \ldots, (i+1) \cdot n_2 - 1]|$ denote the Hamming weight of the i-th RM decoding error block (s.t. $0 \leq i < n_1$), and let $\mathbf{H} \in \mathbb{Z}^{128}$ be the corrected Hadamard output, as defined in Algorithm 4. If $w_i < n_2/4$, the coefficient with the largest absolute value corresponds to the transmitted RM codeword, and*

$$w_i = \frac{n_2}{2} - v_{\max}, \qquad v_{\max} = \max_{0 \leq j < 128} abs(H_j). \tag{3}$$

Proof. Let $a_i \in \mathbb{F}_2^7$ and $b_i \in \mathbb{F}_2$ denote the linear and constant parts of the transmitted RM codeword. For $0 \leq t < mul$ and $0 \leq j < 128$, we have $c_{in_2+128t+j} = \langle a_i, j \rangle + b_i + e'_{in_2+128t+j}$, where additions are performed in $\mathbb{F}_2$. By the definition of the corrected Hadamard transform,

using $x = (1-(-1)^x)/2$ for $x \in \mathbb{F}_2$ and the orthogonality relation $\sum_{j=0}^{127}(-1)^{\langle q,j\rangle} = 128\,\mathbf{1}_{q=0}$, we obtain

$$H_q = \frac{(-1)^{b_i+1}}{2} \sum_{t=0}^{mul-1} \sum_{j=0}^{127} (-1)^{\langle q+a_i,j\rangle + e'_{in_2+128t+j}}. \tag{4}$$

For $q = a_i$, the inner product vanishes, yielding

$$H_{a_i} = (-1)^{b_i+1} \left(\frac{n_2}{2} - w_i\right). \tag{5}$$

For $q \neq a_i$, the orthogonality relation gives

$$H_q = (-1)^{b_i} \sum_{t=0}^{mul-1} \sum_{j=0}^{127} (-1)^{\langle q+a_i,j\rangle} e'_{in_2+128t+j}, \tag{6}$$

and therefore $abs(H_q) \leq w_i$. Consequently, if $w_i < n_2/4$, then

$$abs(H_{a_i}) = \frac{n_2}{2} - w_i > w_i \geq abs(H_q), \qquad q \neq a_i.$$

Thus, the largest absolute Hadamard coefficient corresponds to the transmitted RM codeword, and $v_{\max} = n_2/2 - w_i$, as claimed. □

Proposition 1 establishes a direct relationship between the highest Hadamard peak and the Hamming weight of the corresponding error block. However, the remaining Hadamard coefficients also depend on the error pattern and may therefore contain exploitable information. The following proposition characterizes this dependence.

Proposition 2. *For the i-th RM decoding, let w_i denote the Hamming weight of the corresponding error block and $a_i \in \mathbb{F}_2^7$ the index of the transmitted RM codeword. For any $q \neq a_i$, let $P_{i,q}$ denote the number of errors at positions satisfying $\langle q + a_i, j\rangle = 0$. Then,*

$$abs(H_q) = abs(2 \cdot P_{i,q} - w_i) \leq w_i. \tag{7}$$

Proof. For $q \neq a_i$, the orthogonality of the Walsh–Hadamard basis gives Equation 6. Since $e'_{i\cdot n_2+128t+j} \in \{0,1\}$, only the positions containing an error contribute to this sum. Each error contributes either $+1$ or -1, up to the common sign factor $(-1)^{b_i}$, depending on the value of $\langle q + a_i, j\rangle$. By definition, $P_{i,q}$ counts the errors at positions satisfying $\langle q+a_i, j\rangle = 0$. These errors contribute $+1$ to the sum, whereas the remaining $w_i - P_{i,q}$ errors, for which $\langle q + a_i, j\rangle = 1$, contribute -1. Consequently, $H_q = (-1)^{b_i}\,(P_{i,q} - (w_i - P_{i,q})) = (-1)^{b_i}(2P_{i,q} - w_i)$. Taking absolute values and using $0 \leq P_{i,q} \leq w_i$, we obtain $abs(H_q) = abs(2P_{i,q} - w_i) \leq w_i$, which concludes the proof. □

These results show that the Hadamard coefficients carry different levels of information about the error weight. While the correct coefficient provides a direct relationship with this weight, the remaining coefficients depend on both the error weight and the positions of the errors. Consequently, the leakage of these coefficients may also contribute to error-weight estimation, although their relationship with the targeted quantity is less direct.

These observations motivate investigating the FindPeaks function, which successively manipulates all Hadamard coefficients to identify the one with the largest absolute value. In particular, this maximum is stored in the intermediate variable $v_{\max}$, whose value is directly related to the error weight when the correct peak is selected. We next analyze the implementation of FindPeaks to identify potential sources of side-channel leakage.

Implementation analysis. The C code of the FindPeaks function is given in Figure 2. The variable `peak_abs_value`, which stores the largest absolute value encountered so far, is highlighted in red. As established in Proposition 1, its final value is directly related to the Hamming weight of the corresponding error block when the correct peak is selected. The FindPeaks function iterates over the 128 coefficients produced by the Fast Hadamard Transform. At each iteration, the current coefficient is first converted into its absolute value (Lines 7 to 9) and compared against the current maximum stored in `peak_abs_value` (Lines 10 to 12). The variable is updated if the absolute value of the coefficient is larger than the previous one. As a result, `peak_abs_value` is repeatedly involved in comparisons and updates throughout the loop, providing opportunities for data-dependent leakage.

```
1  int32_t find_peaks(rm_expanded_cdw *transform) {
2      int32_t peak_abs_value = 0; int32_t peak_value = 0; int32_t peak_pos = 0;
3      for (int32_t i = 0; i < 128; i++) {
4          int32_t t = (*transform)[i];
5          int32_t pos_mask = -(t > 0);
6          int32_t absolute = (pos_mask & t) | (~pos_mask & -t);
7          peak_value = absolute > peak_abs_value ? t : peak_value;
8          peak_pos = absolute > peak_abs_value ? i : peak_pos;
9          peak_abs_value = absolute > peak_abs_value ? absolute : peak_abs_value;
10     }
11     peak_pos |= 128 * (peak_value > 0);
12     return peak_pos;
13 }
```

Figure 2: C code of the FindPeaks function from `reed_muller.c` [HQC]. The targeted variable is highlighted in red.

In particular, when the correct peak is selected, its absolute value is stored in the `peak_abs_value` variable and subsequently used in the remaining comparisons. As this value is directly related to the error weight, its manipulation may produce exploitable side-channel leakage. Furthermore, the processing of the other Hadamard coefficients may also contribute to the leakage, although their relationship with the error weight is less direct. This observation is confirmed in practice in Appendix A.

This approach is particularly relevant in our known-ciphertext setting. Unlike the chosen-ciphertext attack, it does not require the message $\mathbf{m}$ to be zero, since the relationship between the correct Hadamard peak and the error weight holds independently of the transmitted message. Moreover, as the attacker knows $\mathbf{m}$, the corresponding Reed–Solomon codeword can be computed, allowing the expected position of the correct Hadamard peak to be determined. The attacker can therefore target the corresponding operations without modifying the encapsulation procedure.

3.3 End-to-End Attack Overview

We have identified two sources of side-channel leakage that can be exploited to estimate the Hamming weight of the error $\mathbf{e}'$, depending on the attacker's capabilities. In the chosen-ciphertext setting (Section 3.1), the attacker forces $\mathbf{m} = \mathbf{0}$ and exploits leakage from the ExpandAndSum function. In the known-ciphertext setting (Section 3.2), the attacker follows the legitimate encapsulation procedure and targets the FindPeaks function, whose intermediate values carry information about the error weight even when the message is

nonzero. Although these attacks target different functions, they share the same end-to-end key-recovery procedure, illustrated in Figure 3.

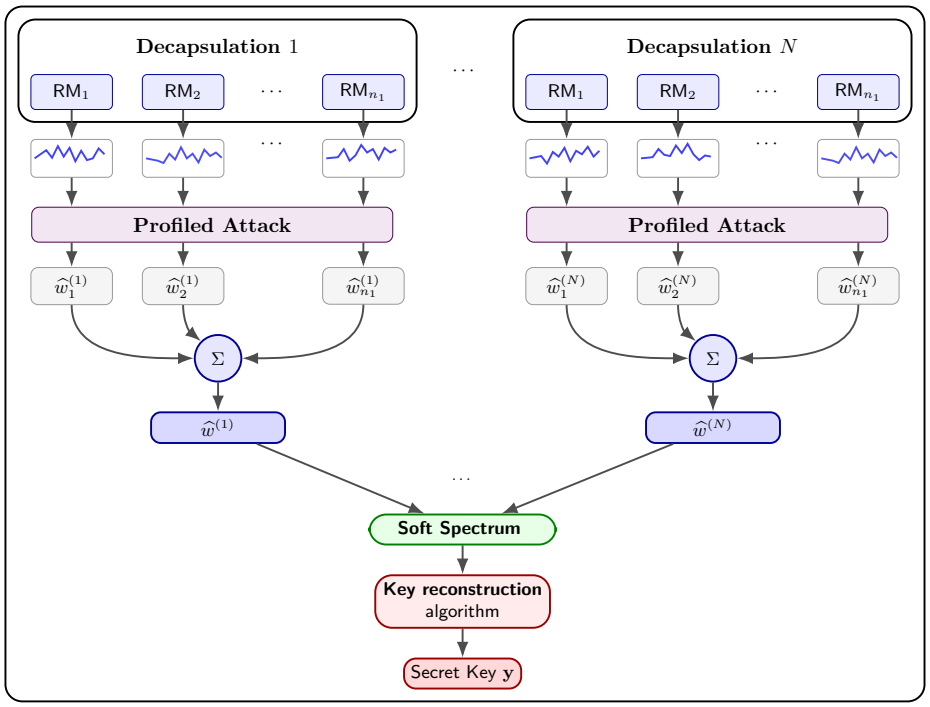


Figure 3: Overview of the end-to-end attack. The profiled attack exploits the weakness identified in Section 3.1 or Section 3.2 depending on the threat model. The notation $\widehat{w}_i^{(j)}$ denotes the estimated Hamming weight of the i-th RM decoding error block of the j-th decapsulation.

For each decapsulation, the n_1 RM decoding operations are targeted separately. In particular, for HQC-1, each decapsulation involves 46 RM decodings, providing 46 corresponding side-channel sub-traces. Depending on the attack setting, these traces capture the leakage of either ExpandAndSum (CCA) or FindPeaks (KCA). A profiled attack (*e.g.* pooled template classifier [CK13] in Section 5) is then applied to each sub-trace to estimate the Hamming weight of the corresponding error block $\mathbf{e}'$. More precisely, the classifier outputs a probability distribution over the possible Hamming weights, from which an estimate is obtained by computing its expectation. For the ℓ-th decapsulation, with $1 \leq \ell \leq N$, let $\widehat{w}_i^{(\ell)}$ denote the estimated Hamming weight of the i-th RM error block, where $0 \leq i < n_1$. The resulting estimates are combined to obtain an estimate of the Hamming weight of the complete error vector $\widehat{w}^{(\ell)} = \sum_{i=0}^{n_1-1} \widehat{w}_i^{(\ell)}$. These steps are repeated for N decapsulations, providing a set of N error-weight estimates. The accuracy of these estimates depends on the attack setting, as the targeted functions and their leakage characteristics differ.

Together with the known sparse vectors $\mathbf{r_1}$, $\mathbf{r_2}$, and $\mathbf{e}$ associated with each decapsulation, these observations are used to estimate the distance spectrum of the secret key $\mathbf{y}$. However, the recovered error weights may be noisy, and the number of observations may be insufficient to determine the distance multiplicities accurately. We therefore rely on a soft distance spectrum, as introduced in [SSV26], which assigns probabilities to the possible multiplicities of each distance. Finally, this soft spectrum is given as input to a key-reconstruction algorithm to recover the secret key $\mathbf{y}$. The next section reviews existing reconstruction techniques and introduces an improved method in Section 4.2 to complete the attack from imperfect error-weight information.

4 Key Recovery from Error Weight

This section describes two methods to recover the secret key from multiple error-weight observations. The idea is to first estimate the distance spectrum of the secret key from these observations and then reconstruct the key from this spectrum. We first present the method proposed in [SSV26] as a starting point before highlighting how our approach differs from it.

4.1 Soft Distance Spectrum Key Recovery

The distance spectrum of a binary vector describes the cyclic distances between its nonzero coordinates, counted with multiplicity. We recall its formal definition from [SSV26].

Definition 1 (Distance Spectrum [SSV26])**.** Let $\mathcal{R} = \mathbb{F}_2[X]/(X^n - 1)$, where n is an odd prime, and let $\mathbf{g} \in \mathcal{R}$.

- The *coordinate distance* between two distinct positions $0 \le i, j < n$ is defined as
$$\mathrm{d}(i, j) = \min\big((i - j) \bmod n,\ (j - i) \bmod n\big).$$
- The *multiplicity* of a distance $1 \le \delta \le n/2$ in $\mathbf{g}$ is defined as
$$\mu_{\mathbf{g}}(\delta) = \left|\{(i, j) \mid g_i = g_j = 1,\ 0 \le i < j < n, \mathrm{d}(i, j) = \delta\}\right|.$$
- The *distance spectrum* of $\mathbf{g} \in \mathcal{R}$ is defined as
$$\mathrm{Sp}(\mathbf{g}) = \big(\mu_{\mathbf{g}}(\delta)\big)_{1 \le \delta \le n/2} \in \mathbb{N}^{\lfloor n/2 \rfloor}.$$

  Each pair of nonzero coordinates contributes to exactly one distance.

Recovering the spectrum. As shown in [SSV26], knowledge of multiple error weights in HQC can lead to the recovery of the distance spectrum of the secret key $\mathbf{y}$. For a given secret key $(\mathbf{x}, \mathbf{y})$ and a set $\mathcal{M} \subset \mathcal{R}^2_{\omega_r} \times \mathcal{R}_{\omega_e}$ of known sparse vectors $(\mathbf{r_1}, \mathbf{r_2}, \mathbf{e})$, the authors introduce the following statistic for each distance $1 \le \delta \le n/2$:

$$F_\delta(\mathcal{M}, (\mathbf{x}, \mathbf{y})) = \frac{\displaystyle\sum_{(\mathbf{r_1}, \mathbf{r_2}, \mathbf{e}) \in \mathcal{M}} \mu_{\mathbf{r_1}}(\delta)\ |\mathbf{r_2 x} - \mathbf{r_1 y} + \mathbf{e}|}{\displaystyle\sum_{(\mathbf{r_1}, \mathbf{r_2}, \mathbf{e}) \in \mathcal{M}} \mu_{\mathbf{r_1}}(\delta)}. \tag{8}$$

The key observation is that F_δ varies depending on the multiplicity of δ in the secret key $\mathbf{y}$. Typically, the higher this multiplicity, the lower the value of F_δ. The multiplicity of each distance can therefore be estimated from the corresponding value of F_δ.

However, when the number of error-weight observations is insufficient, distinguishing between different multiplicities becomes challenging, as their corresponding distributions of F_δ may overlap. The approach proposed in [SSV26] addresses this limitation by exploiting soft information rather than assigning a single multiplicity to each distance. In our attack, the error weights are not directly available but are estimated from side-channel leakage. For the ℓ-th decapsulation, let $\widehat{w}^{(\ell)}$ denote the estimated Hamming weight of $\mathbf{e'}^{(\ell)}$, as introduced in Section 3.3. Using the corresponding known sparse vectors, we construct the empirical statistic

$$\widehat{F}_\delta = \frac{\displaystyle\sum_{\ell=1}^{N} \mu_{\mathbf{r_1}^{(\ell)}}(\delta)\, \widehat{w}^{(\ell)}}{\displaystyle\sum_{\ell=1}^{N} \mu_{\mathbf{r_1}^{(\ell)}}(\delta)}. \tag{9}$$

This provides an estimate of the statistic used for spectrum recovery, subject to the noise introduced by the side-channel estimation and the truncation performed during HQC decryption.

Soft information. Instead of deciding whether a distance belongs to the spectrum, we can assign to each distance a probability distribution over its possible multiplicities based on the value of F_δ. The authors of [SSV26] describe a probabilistic model for F_δ based on Gaussian distributions, one for each possible multiplicity ν. These distributions have theoretical means W_ν, derived in [SSV26, Section 6.2], while their standard deviation σ is determined empirically. Using this model, a multiplicity distribution is derived for each distance and gathered into a soft distance spectrum.

Definition 2 (Soft Distance Spectrum)**.** The soft distance spectrum of a binary vector $\mathbf{g} \in \mathcal{R}_\omega$ is defined as a collection of probability distributions

$$S = (S_{\delta,\nu})_{1 \le \delta \le n/2, \nu \ge 0},$$

where $S_{\delta,\nu}$ is the probability of distance δ to have multiplicity ν. For a given secret key $(\mathbf{x}, \mathbf{y})$ and a set of sparse vectors $(\mathbf{r_1}, \mathbf{r_2}, \mathbf{e}) \in \mathcal{M} \subset \mathcal{R}_{\omega_r}^2 \times \mathcal{R}_{\omega_e}$, this probability is defined for all $1 \le \delta \le n/2$ and for all $\nu \ge 0$ by

$$S_{\delta,\nu} = \frac{\rho_\nu(n,\omega) \cdot \varphi_\nu(F_\delta(\mathcal{M}, (\mathbf{x}, \mathbf{y})))}{\sum_{\gamma \ge 0} \rho_\gamma(n,\omega)\, \varphi_\gamma(F_\delta(\mathcal{M}, (\mathbf{x}, \mathbf{y})))}, \tag{10}$$

where φ_ν is the Gaussian probability density function associated with multiplicity ν (*i.e.* $\varphi_\nu(z) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(z-W_\nu)^2}{2\sigma^2}\right)$) and $\rho_\nu(n,\omega)$ is the prior probability of multiplicity ν for a uniformly sampled vector $\mathbf{g} \in \mathcal{R}_\omega$ (*i.e.* $\rho_\nu(n,\omega) = \Pr[\mu_{\mathbf{g}}(\delta) = \nu] = \frac{\binom{\omega}{\nu}\binom{n-\omega-1}{\omega-\nu-1}}{\binom{n-1}{\omega-1}}$).

In practice, the statistic F_δ is replaced by its estimate $\widehat{F}_\delta$ obtained from side-channel observations, using a probabilistic model consistent with the observed error weights. The resulting soft spectrum therefore accounts for the uncertainty in the estimated distance multiplicities, rather than committing to a single spectrum.

Reconstruction algorithm from [SSV26]. From this soft spectrum, the authors of [SSV26] propose a reconstruction algorithm that attempts to recover the secret key $\mathbf{y}$. The algorithm, given in Algorithm 7, performs a depth-first search that progressively constructs candidate polynomials by adding pairs of monomials. At each step, candidates are selected according to their compatibility with the soft spectrum, measured by the score function $\mathsf{score}(\cdot, S)$. For each candidate, its soft extension $\mathbf{g}^{(S)}$ is computed, ranking the remaining monomials according to their compatibility with the current candidate and the soft spectrum. The algorithm then considers the ω highest-ranked monomials as a potential solution. If this candidate does not satisfy the acceptance criterion, the search continues recursively.

Algorithm 7 Reconstruction from a Soft Distance Spectrum [SSV26].

`KeySearchSoft` : $(\ell, \mathbf{f}, S, \Delta) \mapsto \mathbf{g}$

Require: $\ell \geq 0$, $\mathbf{f} \in \mathcal{R}$, S, $\Delta = (\delta_i)_{i \geq 0}$
Ensure: $\perp$ or $\mathbf{g} \in \mathcal{R}$ such that $|\mathbf{g}| = \omega$
1: $\mathcal{G} \leftarrow \left\{\mathbf{g} = \mathbf{f} \cup (X^s + X^{s+\delta_\ell}),\ 0 \leq s < n,\ \mathsf{score}(\mathbf{g}, S) > T_\ell\right\}$ ▷ T_ℓ determined empirically
2: **for** $\mathbf{g} \in \mathcal{G}$ in decreasing order of $\mathsf{score}(\mathbf{g} + \mathbf{g}^{(S)}[\omega - 1], S)$ **do**
3: **if** "$\mathbf{g}^{(S)}[0{:}\omega]$ is good" **then**
4: **return** $\mathbf{g}^{(S)}[0{:}\omega]$
5: $\mathbf{k} \leftarrow \texttt{KeySearchSoft}(\ell + 1, \mathbf{g}, S, \Delta)$
6: **if** $\mathbf{k} \neq \perp$ **then**
7: **return** $\mathbf{k}$
8: **return** $\perp$

Limitations. A first limitation of this algorithm is that it may return an incorrect solution. Indeed, in Line 3, the acceptance criterion "$\mathbf{g}^{(S)}[0{:}\omega]$ is good" relies on the existence of a sufficiently large gap between the scores of the ω-th and $(\omega + 1)$-th monomials in the soft extension. Although reported in [SSV26] to be an efficient heuristic filter, this criterion does not guarantee that the returned candidate corresponds to the actual secret key. As a result, any candidate must be validated against the public key.

Moreover, a distance spectrum is invariant under cyclic shifts and reversal of the support. Consequently, the secret polynomial can only be determined up to these symmetries. In HQC, the public-key relation $\mathbf{s} = \mathbf{x} + \mathbf{hy}$ can be used to test a candidate $\mathbf{y}$. For a candidate $\widetilde{\mathbf{y}}$, the corresponding polynomial $\widetilde{\mathbf{x}} = \mathbf{s} - \mathbf{h}\widetilde{\mathbf{y}}$ can be computed using the public key. A candidate is compatible with the secret-key constraints only if $|\widetilde{\mathbf{x}}| = \omega$ and $|\widetilde{\mathbf{y}}| = \omega$.

We next introduce a reconstruction algorithm based on simulated annealing that aims to improve the exploitation of soft-spectrum information while incorporating candidate validation into the key-recovery process.

4.2 Simulated Annealing

We describe in this section an algorithm based on simulated annealing to reconstruct the secret key from a soft distance spectrum.

History. In 1983, Kirkpatrick, Gelatt, and Vecchi proposed an optimization method called simulated annealing [KGV83] to approximate the global optimum of a function. It is inspired by the metallurgical process of slowly cooling heated materials. Simulated annealing explores a search space by moving from a current state to a neighboring state. Each state is associated with an objective value that measures the quality of the solution. At each iteration, moves that improve this value are accepted, while worsening moves are accepted with a probability depending on a parameter called the temperature. The temperature decreases throughout the execution, making worsening moves more likely to be accepted at the beginning of the search and increasingly unlikely as the algorithm progresses. Simulated annealing has been primarily used to solve combinatorial optimization problems. It has also found applications in cryptography, for example for the design of symmetric-key primitives [Nik17], or for the cryptanalysis of hash functions [YBL26].

Rationale. The reconstruction algorithm from [SSV26] employs a depth-first search strategy to reconstruct the secret key. At each search depth, a score function is used to rank candidate extensions and determine which branches of the search tree should be explored next. However, two vectors that are close in Hamming distance may correspond

to distant nodes in the search tree, making it difficult to move directly between neighboring candidates.

This motivates the idea of working only with candidates that have the required Hamming weight ω. Starting from such a candidate, we apply elementary transformations that preserve its weight, such as swapping a position inside the support with one outside. This allows the search to visit neighboring solutions without reconstructing them from scratch.

However, a greedy search based on this principle may become trapped in a local optimum. We therefore rely on simulated annealing which is based on the Metropolis-Hastings algorithm. This method probabilistically allows moves towards worse neighbors, based on a score function. These worsening moves become less frequent as the algorithm progresses.

Score function. Our key reconstruction algorithm uses a soft distance spectrum $S = (S_{\delta,\nu})_{1\le\delta\le n/2,\nu\ge 0}$ to recover the secret key $\mathbf{y}$. The model used to derive this soft spectrum from the error-weight observations is the same as the one described in [SSV26], and recalled in Definition 2. We rely on a score function to evaluate the compatibility of candidate polynomials with the soft spectrum.

Definition 3 (Score Function)**.** Given a soft distance spectrum $S = (S_{\delta,\nu})_{1\le\delta\le n/2,\nu\ge 0}$ and a polynomial $\mathbf{g} \in \mathcal{R}$ of Hamming weight at most ω, we define the *score* of $\mathbf{g}$ as

$$\mathsf{score}(\mathbf{g}, S) = \sum_{\delta=1}^{n/2} \log\left(S_{\delta,\mu_{\mathbf{g}}(\delta)}\right). \tag{11}$$

Assuming that the multiplicities associated with different distances are independent, this score corresponds to the sum of their log-probabilities under the soft spectrum. A higher score therefore indicates that the candidate better matches the soft spectrum S. In practice, the independence assumption is only an approximation, since the distance multiplicities of a polynomial are not independent. Designing a score function that does not rely on this independence assumption could be an interesting direction for future work. An important advantage of this score however is that it can be updated efficiently after a support swap.

Algorithm. We now describe the different steps of our algorithm. The complete approach is presented in Algorithm 8. For a support $A \subseteq \{0, \dots, n-1\}$, we denote by $g_A = \sum_{i\in A} X^i \in \mathcal{R}$ its associated polynomial. With a slight abuse of notation, we write $\mathsf{score}(A, S) = \mathsf{score}(g_A, S)$.

Initialization. The algorithm first constructs an initial state corresponding to the support of a candidate secret key, that is, a set of ω positions representing its nonzero coefficients. Because the distance spectrum is invariant under cyclic shifts, we may restrict the search to supports containing position 0. This does not remove any possible distance spectrum, since every nonzero polynomial admits a cyclic shift whose support contains position 0.

The initial support is therefore set to $A = \{0\}$. To complete the remaining support, every position outside A is tested by computing the score difference obtained when adding it to the current candidate:

$$\Delta_p = \mathsf{score}(A \cup \{p\}, S) - \mathsf{score}(A, S). \tag{12}$$

The positions are ranked in decreasing order of Δ_p. The next position is then selected among the k_i highest-ranked candidates, using a probability distribution that favors the best-ranked positions. This process is repeated until the support contains ω positions. The

Algorithm 8 Key reconstruction using simulated annealing

Require: Soft spectrum S, number of restarts R_{SA}, number of iterations I, temperatures T_s, T_f, probability P, parameters k_i, k_e
Ensure: Recovered key $\widehat{\mathbf{y}}$ or $\perp$
$\mathcal{E} \leftarrow \emptyset$, $s_{\max} \leftarrow -\infty$
for $r = 1, \ldots, R_{\text{SA}}$ **do**
  if r is an exploration restart **or** $\mathcal{E} = \emptyset$ **then**
    $A \leftarrow \textsc{Initialize}(S, k_i)$
    explore $\leftarrow$ true
  else
    $A \leftarrow$ a support from $\mathcal{E}$
    explore $\leftarrow$ false
  $s \leftarrow \mathsf{score}(A, S)$
  if $s > s_{\max}$ **then**
    $s_{\max} \leftarrow s$
    $\mathcal{E} \leftarrow \textsc{AddToElite}(\mathcal{E}, A, s)$
    $\widehat{\mathbf{y}} \leftarrow \textsc{IsKey}(A)$
    if $\widehat{\mathbf{y}} \neq \perp$ **then return** $\widehat{\mathbf{y}}$
  for $i = 1, \ldots, I$ **do**
    $T \leftarrow T_s(T_f/T_s)^{i/I}$
    if *explore* **then**
      $A' \leftarrow \textsc{Swap}(A)$
    else
      $A' \leftarrow \textsc{GuidedSwap}(A, S, P, k_e)$
    $\Delta \leftarrow \mathsf{score}(A', S) - s$
    if $\Delta \geq 0$ **or** $\text{rand}(0,1) < e^{\Delta/T}$ **then**
      $A \leftarrow A'$
      $s \leftarrow s + \Delta$
      if $s > s_{\max}$ **then**
        $s_{\max} \leftarrow s$
        $\mathcal{E} \leftarrow \textsc{AddToElite}(\mathcal{E}, A, s)$
        $\widehat{\mathbf{y}} \leftarrow \textsc{IsKey}(A)$
        if $\widehat{\mathbf{y}} \neq \perp$ **then return** $\widehat{\mathbf{y}}$
return $\perp$

Algorithm 9 Initialize

Require: Soft spectrum S, parameter k
Ensure: Candidate support A (s.t $|A| = \omega$)
$A \leftarrow \{0\}$
while $|A| < \omega$ **do**
  for $p \in \{0, \ldots, n-1\} \setminus A$ **do**
    $\Delta_p \leftarrow \mathsf{score}(A \cup \{p\}, S) - \mathsf{score}(A, S)$
  Rank positions p by decreasing Δ_p
  Sample $j \in \{0, \ldots, k-1\}$ with $\Pr[j] \propto e^{-j/5}$
  $A \leftarrow A \cup \{p_j\}$
return A

Algorithm 10 GuidedSwap

Require: Support A, soft spectrum S, probability P, number of trials k
Ensure: Neighboring support A'
$q \xleftarrow{\$} A$
if $\text{rand}(0,1) < P$ **then**
  $p \xleftarrow{\$} \{0, \ldots, n-1\} \setminus A$
else
  $p_1, \ldots, p_k \xleftarrow{\$} \{0, \ldots, n-1\} \setminus A$
  $p \leftarrow \arg\max_{p_i} \mathsf{score}((A \setminus \{q\}) \cup \{p_i\}, S)$
$A' \leftarrow (A \setminus \{q\}) \cup \{p\}$
return A'

Initialize procedure is detailed in Algorithm 9. The resulting candidate is then used as the initial state of the simulated annealing procedure.

Simulated annealing. Once the initial state has been constructed, the simulated annealing procedure begins. At each iteration, one position inside the candidate support A is randomly swapped with one outside to generate a neighboring candidate A', using the Swap procedure. The score difference is computed as

$$\Delta = \mathsf{score}(A', S) - \mathsf{score}(A, S). \tag{13}$$

If $\Delta \geq 0$, the swap is automatically accepted. Otherwise, it is accepted with probability $\Pr[\text{accept}] = \exp\left(\frac{\Delta}{T}\right)$, where $T > 0$ denotes the current temperature. The temperature decreases geometrically from a started temperature T_s to a final temperature T_f over I iterations:

$$T_i = T_s \left(\frac{T_f}{T_s}\right)^{\frac{i}{I}}, \qquad 1 \leq i \leq I. \tag{14}$$

As the algorithm progresses, moves toward candidates with lower scores become less likely to be accepted.

When a candidate achieves a higher score than the best score encountered so far, the candidate and its score are saved. We also test the candidate against the public key to determine whether it corresponds to a valid secret key, using the IsKey procedure. Recall

that the distance spectrum is invariant under cyclic shifts and reversal. Consequently, a candidate support obtained from the soft spectrum may correspond to a shifted or reversed version of the actual secret key. To address this ambiguity, IsKey considers the cyclic shifts and reversals of the candidate and verifies their compatibility with the public key. In particular, for a candidate $\widetilde{\mathbf{y}}$ (as well as for all its cyclic shifts and their reversals), we compute $\widetilde{\mathbf{x}} = \mathbf{s} - \mathbf{h}\widetilde{\mathbf{y}}$ and check the secret-key Hamming-weight constraints. If a valid candidate is found, the algorithm returns the corresponding key in its verified orientation. Otherwise, the search continues.

Restarts. The simulated annealing procedure is repeated several times using different initial states. Each repetition is called a restart. We distinguish between two types of restarts: *exploration restarts* and *elite restarts*. Exploration restarts begin with a new state generated using the initialization procedure described above. They allow the algorithm to explore different regions of the search space. Elite restarts begin from a state encountered during the simulated annealing process that achieved a high score. Such states, together with their scores, are stored in an elite pool $\mathcal{E}$ through the AddToElite procedure. The goal of elite restarts is to further explore regions containing promising candidates. An elite restart uses a different swap procedure from exploration restarts. Indeed, the position to add to the current support is not always selected uniformly at random. The procedure, denoted GuidedSwap, first randomly selects a position inside the support to remove. Then, with probability P, the new position is selected uniformly among those outside the support. Otherwise, the procedure tests k_e candidate positions outside the support and chooses the one producing the highest score after the swap. This strategy combines random exploration with a preference for promising neighboring solutions. The GuidedSwap process is detailed in Algorithm 10.

With this approach, we address the limitations discussed in the previous section. In particular, the IsKey procedure verifies candidate keys against the public key, rather than relying solely on a heuristic score-based acceptance criterion. Moreover, by testing cyclic shifts and reversals of the reconstructed support, the procedure can identify a candidate in the correct orientation instead of returning an arbitrary representative of its distance spectrum. Finally, the use of simulated annealing allows the search to explore neighboring fixed-weight candidates while escaping local optima through probabilistic acceptance of worsening moves. To assess the effectiveness of our method, we compare it experimentally with the reconstruction algorithm from [SSV26] in the next section.

4.3 Simulation-based Validation

In this section, we present numerical results for Algorithm 8 and compare its performance with Algorithm 7. To do so, we rely on simulations following the same protocol as in [SSV26] to construct a soft distance spectrum. All experiments, including those involving Algorithm 7, were performed on a machine equipped with an Intel Xeon w3-2423 processor with a maximum clock frequency of 4 GHz. The C implementations were compiled with GCC version 13.3.0 using the `-O3` optimization flag. The C code is available at https://github.com/m-salom/sca-spectrum.

Protocol. The experimental protocol is the following. For a given secret key $(\mathbf{x}, \mathbf{y})$, we generate a collection $\mathcal{M} = \left\{\left(\mathbf{r_1}^{(\ell)}, \mathbf{r_2}^{(\ell)}, \mathbf{e}^{(\ell)}\right)\right\}_{\ell=1}^{N} \subseteq \mathcal{R}_{\omega_r}^2 \times \mathcal{R}_{\omega_e}$, where $N = |\mathcal{M}|$ is the number of error-weight observations. We then proceed as follows:

1. Compute $F_\delta(\mathcal{M}, (\mathbf{x}, \mathbf{y}))$ for all $1 \leq \delta \leq n/2$ using Equation 8;
2. Compute $S_{\delta,\nu}$ for all $1 \leq \delta \leq n/2$ and $0 \leq \nu \leq \omega - 1$ using Equation 10. In practice, ν rarely exceeds 20, so we restrict its range to $0 \leq \nu \leq 20$.

Remark 1. We use a centered model and centered observations to improve the accuracy of the estimated distance multiplicities, as explained in [SSV26, Section 6.2].

The resulting soft spectrum $S = (S_{\delta,\nu})_{1\leq\delta\leq n/2, 0\leq\nu\leq\omega-1}$ is given as input to both reconstruction algorithms. We emphasize that these simulations use the Hamming weight of the complete error polynomial $\mathbf{e}_{\text{full}} = \mathbf{r_2}\mathbf{x} - \mathbf{r_1}\mathbf{y} + \mathbf{e}$, as defined in Equation 8. In contrast, the side-channel attacks presented in Section 3.3 estimate the Hamming weight of its truncated version $\mathbf{e}' = \text{Truncate}(\mathbf{e}_{\text{full}}, n_1 n_2)$. The simulations therefore allow us to evaluate the reconstruction algorithms independently of the physical leakage acquisition.

Parameter selection. The parameters of our simulated annealing algorithm were empirically chosen by evaluating different configurations. Table 2 reports the values considered for each parameter and the ones we selected which yielded the best results during our investigations. We stress that the selected values may not constitute the optimal configuration, and that the results could potentially be improved by choosing another set of parameters. In particular, the probability P controlling the random moves in GuidedSwap was fixed to 0.8, while the remaining parameters were selected empirically. These values are used throughout all subsequent experiments in this paper (including Section 5).

Table 2: Tested and selected parameters for the simulated annealing algorithm.

Parameter	Tested values	Selected value
Number of restarts R	$20, 25, 30$	30
Number of iterations I	$200\,000, 1\,000\,000, 2\,000\,000$	200 000
Start temperature T_s	$1, 2, 10$	2
Final temperature T_f	$0.0001, 0.001, 0.01, 0.1$	0.0001
Initialization top-k_i	$10, 30, 50, 100$	30
Guided swap trials k_e	$500, 1\,000$	500
Guided swap probability P	0.8	0.8

Comparison with existing reconstruction. Table 3 reports the minimum number of error-weight observations and the average running time required to recover the key with Algorithms 7 and 8 for HQC-1 parameters. For each target success probability p, the reported number of observations corresponds to the smallest evaluated value of $N = |\mathcal{M}|$ achieving the desired empirical success rate over 100 independent executions.

Table 3: Comparison of the near-minimum number of error-weight observations and average running times required by Algorithms 7 and 8 to achieve an empirical success rate of at least p for HQC-1 parameters. Observation counts are determined with a resolution of 10 000 for Algorithm 7 and 1 000 for Algorithm 8. Results are measured over 100 executions.

	$p = 0.5$		$p = 0.95$	
	[SSV26]	Ours	[SSV26]	Ours
$N = \lvert\mathcal{M}\rvert$	110 000	25 000	190 000	33 000
Avg. Running Time (s)	251	39	85	35

Table 3 shows that our simulated annealing algorithm requires significantly fewer observations than the algorithm proposed in [SSV26] to reach the same success probability. For a success probability $p = 0.5$, Algorithm 7 requires 110 000 observations, while our method requires only 25 000, corresponding to a reduction by a factor of 4.4. For a higher success probability $p = 0.95$, this reduction is even more significant. Indeed, the number of observations is reduced by a factor of approximately 5.8, from 190 000 required by the

original method to 33 000 with our approach. These results show that our approach is particularly effective at achieving high key-recovery success probabilities with a limited number of observations. In particular, increasing the success probability from 0.5 to 0.95 requires only 8 000 additional observations with our algorithm, compared to 80 000 for [SSV26].

In addition to reducing the number of required observations, our algorithm also reduces the average execution time. For $p = 0.5$, the average running time decreases from 251 to 39 seconds, corresponding to a speedup of approximatelyp 6.4. For $p = 0.95$, it decreases from 85 to 35 seconds. The average execution time of simulated annealing remains relatively stable across the tested numbers of observations. In our experiments, a successful candidate is often found during the early elite restarts. Indeed, we chose to favor the number of restarts over the number of iterations. A large number of iterations can be time-consuming without necessarily improving the quality of the solution. In contrast, increasing the number of restarts allows the algorithm to explore different regions of the search space and increases the chances of starting from a promising candidate. Such candidates typically achieve higher scores and can be further improved through elite restarts, whereas candidates with poor scores may require significantly more iterations to reach a valid solution.

Noisy Information. To better reflect the information an attacker could obtain through a side-channel attack, we also consider noisy error-weight observations. More precisely, for the ℓ-th observation, we consider $z^{(\ell)} = \left|\mathbf{r_2}^{(\ell)}\mathbf{x} - \mathbf{r_1}^{(\ell)}\mathbf{y} + \mathbf{e}^{(\ell)}\right| + \varepsilon^{(\ell)}$, where the noise variables are independently sampled according to $\varepsilon^{(\ell)} \sim \mathcal{N}(0, \sigma^2_{|\mathbf{e}_{\text{full}}|})$. Here, $\sigma_{|\mathbf{e}_{\text{full}}|}$ denotes the standard deviation of the noise added to each error-weight observation. These noisy observations are used in place of the exact error weights to construct the soft distance spectrum.

We evaluate the impact of different noise levels on the number of observations required to recover the key with a success probability of $p = 0.5$, as well as on the average execution time of both Algorithms 7 and 8.

Table 4: Comparison of the near-minimum number of error-weight observations and average running times required by Algorithms 7 and 8 to achieve an empirical success rate of at least $p = 0.5$ under different noise levels for HQC-1 parameters. Observation counts are determined with a resolution of 10 000 for Algorithm 7 and 1 000 for Algorithm 8. Results are measured over 100 executions.

	$N = \lvert\mathcal{M}\rvert$		Avg. Running Time (s)	
	[SSV26]	Ours	[SSV26]	Ours
$\sigma_{\lvert\mathbf{e}_{\text{full}}\rvert} = 10$	110 000	26 000	264	45
$\sigma_{\lvert\mathbf{e}_{\text{full}}\rvert} = 30$	140 000	32 000	282	38
$\sigma_{\lvert\mathbf{e}_{\text{full}}\rvert} = 50$	200 000	44 000	319	40

Table 4 shows that our simulated annealing algorithm requires fewer observations than the algorithm proposed in [SSV26], even in the presence of noise. For a standard deviation of $\sigma_{|\mathbf{e}_{\text{full}}|} = 10$, our algorithm requires only 26 000 observations, compared to 110 000 for [SSV26], corresponding to a reduction by a factor of approximately 4.2. For higher noise levels, this advantage is maintained. When $\sigma_{|\mathbf{e}_{\text{full}}|} = 30$, the number of required observations decreases from 140 000 to 32 000, corresponding to a factor of approximately 4.4. Similarly, for $\sigma_{|\mathbf{e}_{\text{full}}|} = 50$, our approach requires only 44 000 observations instead of 200 000, reducing the required number by a factor of approximately 4.5. As expected, increasing the noise level leads to an increase in the number of observations required by both algorithms. However, our approach maintains an advantage across all tested noise levels. Hence, the proposed reconstruction method remains effective even when the available error-weight information is noisy.

Furthermore, our algorithm also achieves a lower average execution time. While the execution time of [SSV26] increases from 264 to 319 seconds as the noise level increases, our algorithm maintains an average execution time between 38 and 45 seconds. These results show that our approach reduces both the number of required observations and the average execution time of the key reconstruction algorithm, even in the presence of noisy information. Having evaluated the effectiveness of our method through simulations, we now extend our evaluation to a practical side-channel attack to further validate our approach.

5 Experiments

In this section, we experimentally evaluate both our chosen-ciphertext and known-ciphertext attacks. We begin with a description of our experimental setup and a recap of the attack procedure. We then present the results of both our CCA and KCA settings and compare the performance of our simulated annealing method with the reconstruction algorithm from [SSV26]. The code used to obtain the results in this section is available at https://github.com/m-salom/sca-spectrum.

5.1 Setup

Experimental setup. We acquired side-channel traces using a ChipWhisperer Husky with a CW313 board and an ATSAM4S2A microcontroller target based on a 32-bit ARM Cortex-M4 core. The target was clocked at 7.37 MHz. Due to the length of the traces, acquisitions were performed in streaming mode at a sampling rate of 7.37 MS/s. We targeted the HQC-1 parameter set, for which each decapsulation involves $n_1 = 46$ independent RM decoding operations of $n_2 = 384$ bits. The reference implementation of HQC [HQC, Version 5] was compiled using the `arm-none-eabi-gcc` compiler with the `-O3` optimization flag.

Attack recap. A pooled template attack [CK13] has been used in the experiments. This attack is divided into two phases: profiling and attack. During the profiling phase, we acquired traces targeting the ExpandAndSum function in the CCA setting and the FindPeaks function in the KCA setting. For each decapsulation, the decoder performs 46 independent RM decoding operations, providing 46 corresponding side-channel sub-traces. Consequently, a single HQC decapsulation provides 46 profiling sub-traces, allowing us to reduce the number of decapsulation acquisitions needed to construct the profiling dataset. The traces were aligned and normalized to facilitate further processing and improve the accuracy of the leakage model.

Before conducting the attacks, we computed the signal-to-noise ratio (SNR) [MOP07] associated with the Hamming weight of each 384-bit RM error block for the ExpandAndSum and FindPeaks functions. The corresponding results are presented in Appendix A and confirm the analysis conducted in Section 3 as well as the presence of leakages. A pooled template model [CK13] was then trained on the profiling traces and used as a classifier to obtain a probability distribution over the targeted Hamming weight.

In the following, an *attack trace* corresponds to the execution of one complete HQC decapsulation, including its 46 RM decoding operations. Each attack trace was divided into 46 sub-traces to match the structure used during profiling. For the ℓ-th decapsulation, let $p_i^{(\ell)}(w)$ denote the probability assigned by the classifier to the Hamming weight w of the i-th RM error block, with $0 \leq i < n_1$. For each sub-trace, we compute the expected Hamming weight as $\widehat{w}_i^{(\ell)} = \sum_{w=0}^{n_2} w\, p_i^{(\ell)}(w)$. The estimated Hamming weight of the complete error vector $\mathbf{e}'^{(\ell)}$ is finally obtained by summing the expectations over

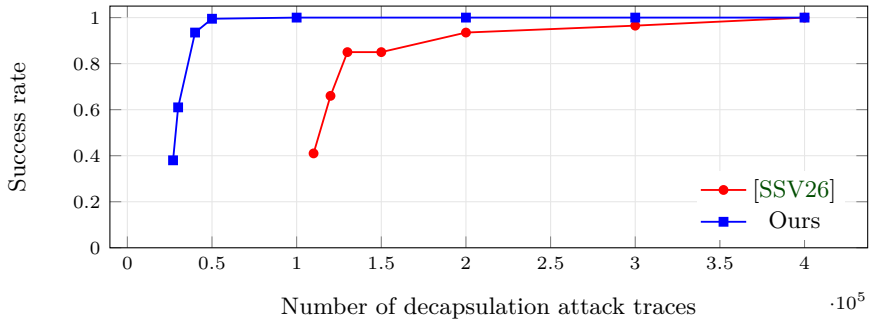


Figure 4: Comparison of the CCA key-recovery success rates as a function of the number of decapsulation traces. Results are measured over 200 trials per trace count using Algorithm 8 (blue) and Algorithm 7 (red).

the 46 sub-traces $\widehat{w}^{(\ell)} = \sum_{i=0}^{n_1-1} \widehat{w}_i^{(\ell)}$. These estimates, together with the corresponding known encryption vectors, are used to construct the soft distance spectrum introduced in Section 4.

Table 5 summarizes the number of traces acquired during the profiling and attack phases for both attack settings.

Table 5: Number of decapsulations profiling and attack traces for both settings. Each decapsulation provides 46 RM decoding sub-traces.

	Number of profiling traces	Number of attack traces
CCA	10 000	410 000
KCA	164 000	1 394 000

For the CCA scenario, a total of 460 000 profiling sub-traces were acquired with varying secret keys, corresponding to 10 000 decapsulations. For the KCA scenario, a total of 7 544 000 profiling sub-traces were acquired with varying secret keys, corresponding to 164 000 decapsulations.

For each setting, we performed 200 attack trials per number of traces. Each trial consists in randomly selecting a given number of attack traces from the acquired dataset and attempting to recover the secret key. We use the empirical key-recovery success rate to measure the performance of the attacks, as presented in the following two subsections.

5.2 Chosen-Ciphertext End-to-End Attack

In this section, we compare the results of the chosen-ciphertext attack using the reconstruction algorithm from [SSV26] (Algorithm 7) and our simulated annealing algorithm (Algorithm 8). The attack exploits side-channel leakage from the ExpandAndSum function to estimate the Hamming weight of the error vector $\mathbf{e}'$. These estimates are then used to construct a soft distance spectrum and recover the secret key.

The success rates of both approaches are shown in Figure 4. A trial is considered successful when the reconstruction algorithm recovers a secret key that passes the verification procedure.

Figure 4 shows that our simulated annealing algorithm requires significantly fewer attack traces to achieve comparable success rates. With 27 000 traces, our algorithm achieves a success rate of 0.38, while the algorithm from [SSV26] requires 110 000 traces to reach a similar success rate of 0.41. This corresponds to a reduction in the number of required traces by a factor of approximately 4. Similarly, our approach achieves a success

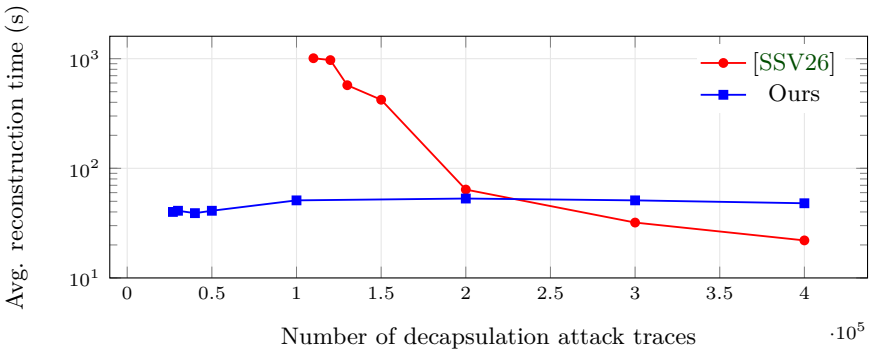


Figure 5: Comparison of the average reconstruction times of Algorithm 8 (blue) and Algorithm 7 (red) as a function of the number of decapsulation traces. Results are measured over 200 trials per trace count.

rate of 0.935 with 40 000 traces, whereas the algorithm from [SSV26] requires 200 000 traces to reach the same success rate. In this case, the number of required traces is reduced by a factor of 5. Moreover, our algorithm achieves an empirical success rate of 1 with 100 000 traces, compared to 400 000 for [SSV26]. These results demonstrate that the improvement observed in our simulations also applies to a practical side-channel attack.

The average execution times of both reconstruction algorithms are presented in Figure 5.

Our algorithm also significantly reduces the reconstruction time when the number of attack traces is limited. Algorithm 7 reaches an average execution time of more than 1 000 seconds with 110 000 traces, corresponding to a success rate of approximately 0.4. In comparison, Algorithm 8 recovers the key in approximately 40 seconds with 27 000 traces, achieving a similar success rate. The execution time of our simulated annealing algorithm remains relatively stable, ranging from 39 to 53 seconds across the tested numbers of attack traces. This behavior is consistent with the restart strategy discussed in Section 4.3. Interestingly, the execution time of Algorithm 7 decreases substantially as the number of traces increases. A more accurate soft spectrum can guide the reconstruction more effectively, reducing the amount of search required. Consequently, although Algorithm 7 can become faster when sufficient observations are available, our method provides a considerable computational advantage when the available error-weight information, and thus side-channel traces, is limited.

Overall, these experimental results confirm that our approach significantly reduces the number of side-channel traces required for key recovery while improving reconstruction efficiency in the low-observation regime.

5.3 Known-Ciphertext End-to-End Attack

From physical leakage to a statistical weight oracle. In the known-ciphertext attack, we exploit side-channel leakage from the FindPeaks function to estimate the Hamming weight of the error vector $\mathbf{e}'$. Unlike the CCA setting, the attacker cannot impose a zero message. Consequently, the estimated error weights are less directly related to the intermediate values manipulated during RM decoding. With the KCA targeting FindPeaks, the error weights estimated by the pooled template classifier [CK13] exhibit a Pearson correlation of $\rho = 0.21$ with the actual error weights. Moreover, these estimates are affected by noise and may exhibit a systematic bias. The original parameters of the Gaussian distributions used to construct the soft spectrum are therefore no longer well suited to our setting, and the model must be adapted. We hence estimate a mapping between the predicted and actual error weights. For the ℓ-th decapsulation, let $w^{(\ell)} = \left|\mathbf{e}'^{(\ell)}\right|$ denote the actual error weight,

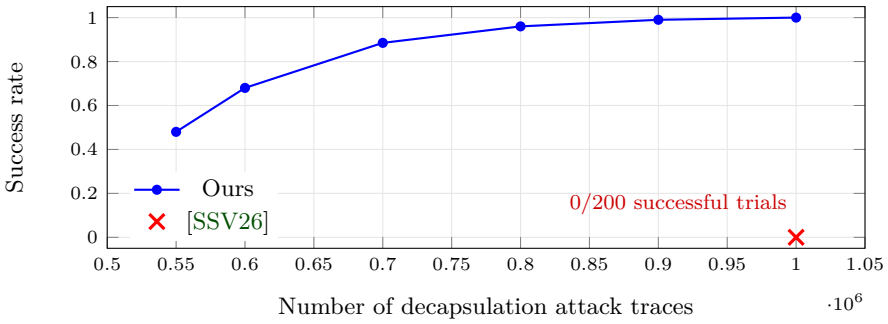


Figure 6: Comparison of the KCA key-recovery success rates of Algorithm 8 (blue curve) and Algorithm 7 from [SSV26] (red cross). Our algorithm achieves an empirical success rate of 100% using 1 000 000 decapsulation traces, whereas Algorithm 7 did not recover the key in any of the 200 trials performed with 1 000 000 traces.

and let $\widehat{w}^{(\ell)}$ be its estimate obtained from the pooled template classifier. We perform an affine calibration on a set of validation traces by estimating a and b such that

$$\widehat{w}^{(\ell)} = a\, w^{(\ell)} + b + \varepsilon^{(\ell)}, \tag{15}$$

where $\varepsilon^{(\ell)}$ represents the residual estimation noise. Using the estimated parameters a and b, the Gaussian model used to construct the soft distance spectrum can be adapted to the error weights recovered by the side-channel attack. Recall that, in the original model, the distance statistic F_δ is approximated by a Gaussian distribution whose mean depends on the distance multiplicity ν. We denote this mean by W_ν and assume a variance of the form

$$\mathrm{Var}(F_\delta \mid \mu_{\mathbf{y}}(\delta) = \nu) \approx \frac{\sigma_0^2}{N},$$

where σ_0 is an empirically determined parameter and N is the number of attack traces. Under the affine calibration model, the mean becomes $\widetilde{W}_\nu = aW_\nu + b$. Furthermore, accounting for the noise affecting the predicted error weights, we use the following variance approximation for the aggregated distance statistic $\widetilde{\sigma}^2 = \frac{a^2\sigma_0^2}{\rho^2 N}$. This approximation assumes that the residual estimation noise is independent of the underlying error weight and that the variance of the aggregated statistic scales as $1/N$. The calibrated Gaussian distributions are then used to compute the soft distance spectrum from the predicted error weights.

Results. Figure 6 presents the success rate of our simulated annealing algorithm as a function of the number of attack traces in the KCA setting. It shows that the success rate increases with the number of attack traces.

With 550 000 traces, our algorithm achieves a success rate of 0.48. This increases to 0.885 with 700 000 traces and reaches 0.96 with 800 000 traces. Finally, with 1 000 000 traces, the correct secret key is recovered in all 200 experimental trials, corresponding to an empirical success rate of 1. These results demonstrate that our approach can successfully recover the secret key from noisy error-weight estimates obtained through a practical side-channel attack.

Unlike in the previous attack scenario, we cannot directly compare the full success-rate curves of our approach and Algorithm 7 from [SSV26], as the available attack traces were insufficient to achieve successful key recovery using the latter algorithm. Indeed, Algorithm 7 was evaluated in 200 trials using 1 000 000 and 1 300 000 attack traces, without recovering the correct secret key in any trial. This result further highlights the advantage

of our simulated annealing approach when the available soft-spectrum information is particularly noisy.

Table 6 reports the average execution time of our simulated annealing algorithm for different numbers of attack traces. The average execution time remains relatively stable, ranging from 36 to 43 seconds over the tested numbers of attack traces. In particular, increasing the number of traces from 550 000 to 1 000 000 improves the empirical success rate from 0.48 to 1, while the average execution time increases by only 7 seconds. This behavior is consistent with the results obtained in Section 5.2 and can be explained by the restart strategy discussed in Section 4.3.

Table 6: Average reconstruction time of Algorithm 8 for different numbers of KCA traces, measured over 200 trials per trace count.

Number of traces	550 000	600 000	700 000	800 000	900 000	1 000 000
Avg. execution time (s)	36	41	39	39	41	43

Compared to the previous chosen-ciphertext attack scenario, the known-ciphertext attack requires significantly more traces to achieve comparable success rates. While the CCA achieves an empirical success rate of 1 with 100 000 traces, the KCA requires 1 000 000 traces to reach the same success rate. This difference reflects the distinct leakage sources and attack conditions. In particular, the KCA targets a less direct relationship between the intermediate values manipulated by FindPeaks and the error weight, whereas the CCA directly exploits the error-dependent computations performed by ExpandAndSum. Nevertheless, these results demonstrate that our approach remains effective in the more constrained KCA setting, where ciphertexts are generated through the legitimate encapsulation procedure.

6 Discussion

6.1 Masking countermeasure

Among the different countermeasures proposed to protect cryptographic implementations against side-channel attacks, masking remains one of the most widely adopted techniques. The motivation behind masking is that physical leakage, such as power consumption or electromagnetic emissions, can depend on the data manipulated by the device. Masking mitigates this dependency by replacing sensitive intermediate values with randomized representations obtained through secret sharing. More precisely, let $d > 1$ be an integer. A $(d-1)$th-order masking scheme represents each sensitive value $z \in \mathbb{K}$ by d shares $(z_1, \ldots, z_d) \in \mathbb{K}^d$, where $d-1$ shares are sampled independently and uniformly at random, while the remaining share is computed such that $z = z_1 \star z_2 \star \cdots \star z_d$, for an operation $\star$ under which $(\mathbb{K}, \star)$ forms a finite group. This construction ensures that any subset containing at most $d-1$ shares is statistically independent of the sensitive value z. While some linear operations can be performed independently on each share, nonlinear operations generally require dedicated algorithms, called *gadgets*, which preserve correctness and the desired security order. In [GSA+25], the authors proposed a fully masked implementation of HQC at arbitrary masking orders.[1] In particular, their implementation relies on gadgets satisfying the Multiple-Inputs Multiple-Outputs Strong Non-Interference (MIMO-SNI) security notion [CS20], allowing the security of their composition to be established in the probing model. This provides formal security guarantees against adversaries observing at most t intermediate values, for $t < d$, under the assumptions of the model.

[1] Implementation available at https://github.com/maximespy/masked_HQC_implementation.

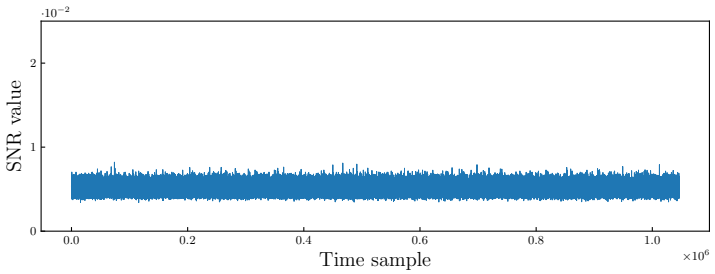


Figure 7: SNR measured during the execution of masked FindPeaks for 70 000 sub-traces, using the Hamming weight of the RM error block as the targeted variable.

Protection against our attacks. In the context of our attacks, masking is particularly relevant since both the CCA and KCA exploit intermediate values related to the Hamming weight of the decryption error $\mathbf{e}'$. In the CCA setting, the attacker targets the ExpandAndSum function, whose intermediate computations directly depend on the error bits when $\mathbf{m} = \mathbf{0}$. Masking prevents these sensitive values from being manipulated in an unprotected representation, thereby mitigating the first-order leakage exploited by our attack. Similarly, in the KCA setting, the attacker exploits the intermediate values manipulated by the FindPeaks function. As established in Proposition 1, the correct Hadamard peak is directly related to the Hamming weight of the corresponding RM error block. Consequently, masking must also protect the Hadamard coefficients and the intermediate computations performed during peak detection. The implementation proposed in [GSA+25] addresses these requirements by providing masked versions of the RM decoding operations, including ExpandAndSum, Hadamard, and FindPeaks.

Leakage evaluation. To illustrate the effectiveness of masking against the leakage targeted by our KCA, we consider the SNR associated with the Hamming weight of the RM error block during the execution of a masked FindPeaks implementation at order 1 (*i.e.* $d = 2$). Figure 7 presents the corresponding SNR result. As shown in Figure 7, the SNR remains relatively low and does not exhibit any prominent peak over the observed execution window in opposition to the unprotected implementation (see Figure 10). This suggests that masking substantially reduces the first-order leakage related to the error weight that is exploited by our KCA. Although masking provides an effective countermeasure against the identified leakage, it introduces significant performance overhead. Further investigations could therefore focus on developing masking-friendly implementations of HQC's error-correction procedure, particularly the RM decoder, to reduce this overhead while preserving the desired security guarantees.

Overall, these observations highlight that protecting HQC against our attacks requires securing not only the secret-key operations but also the intermediate computations performed during error correction. In particular, preventing the leakage of the error weight is essential, as this information alone can be sufficient to enable secret-key recovery.

6.2 Reed–Solomon

While our attacks target the Reed–Muller (RM) decoder, the Reed–Solomon (RS) decoder may also carry information about the Hamming weight of the error $\mathbf{e}'$. In HQC, the duplicated RM code is the inner code, and its decoded symbols form the input to the outer RS code. A higher Hamming weight of $\mathbf{e}'$ may induce more decoding errors in the inner RM blocks, which in turn produce more erroneous symbols at the input of the RS decoder.

We investigate this relationship using the probabilistic model from [BDM+26].

Notations. Let $\mathbf{e}_{\text{full}} = \mathbf{x}\mathbf{r_2} - \mathbf{y}\mathbf{r_1} + \mathbf{e} \in \mathcal{R}$ denote the error before truncation, with $\mathbf{e}' = \text{Truncate}(\mathbf{e}_{\text{full}}, n_1 n_2)$. Let $\mathbf{c}_{\text{RS}} \in \mathbb{F}_{256}^{n_1}$ be the transmitted RS codeword and $\widehat{\mathbf{c}}_{\text{RS}}$ the output of the inner RM decoding. We consider the following quantities:

$$w_{\text{RM}} = |\mathbf{e}'|\,, \quad w_{\text{RS}} = \sum_{i=0}^{n_1-1} \mathbf{1}_{\widehat{c}_{\text{RS},i} \neq c_{\text{RS},i}}. \tag{16}$$

Probabilistic model [BDM+26, Model 2]. For two binary vectors of respective weights ω and ω_r, Model 2 uses hypergeometric counts with parameters (n, ω, ω_r). The dependencies between coefficients are taken into account by conditioning on the fixed sum $p = \omega\omega_r$ of the product in $\mathbb{Z}[X]/(X^n - 1)$. Let

$$h_j = \frac{\binom{\omega}{j}\binom{n-\omega}{\omega_r - j}}{\binom{n}{\omega_r}}, \qquad P(U,V) = \sum_j h_j U^j V^{j \bmod 2}.$$

The modeled Hamming-weight distribution of a binary product $\mathbf{t} \in \mathcal{R}$ is $\Pr(|\mathbf{t}| = s) = \frac{[U^p V^s] P(U,V)^n}{[U^p] P(U,1)^n}$ where $[U^a V^b]$ denotes coefficient extraction. For two independent product vectors $\mathbf{t}^{(1)}$ and $\mathbf{t}^{(2)}$, and an independent uniformly sampled $\mathbf{e} \in \mathcal{R}_{\omega_e}$, we consider $\mathbf{e}_{\text{full}} = \mathbf{t}^{(1)} + \mathbf{t}^{(2)} + \mathbf{e}$. We denote by $\pi_w = \Pr(w_{\text{RM}} = w)$ the resulting modeled distribution of the error weight. Conditionally on $w_{\text{RM}} = w$, we assume that the error positions are uniformly distributed among the n coefficients.

The first $n_1 n_2$ positions are partitioned into n_1 RM blocks of length n_2. The remaining $n - n_1 n_2$ positions are discarded by truncation. For an RM block containing j errors, we approximate the probability of incorrect decoding using

$$q_j = \frac{\Gamma^{\text{ub}}_{\text{RM},j}}{\binom{n_2}{j}},$$

where $\Gamma^{\text{ub}}_{\text{RM},j}$ is obtained from [BDM+26, Proposition 3] with zero quantization error ($\hat{e} = 0$). We define

$$B(z) = \sum_{j=0}^{n_2} \Gamma^{\text{ub}}_{\text{RM},j} z^j, \quad G(z) = (1+z)^{n_2} - B(z). \tag{17}$$

The polynomials B and G enumerate, respectively, the modeled incorrectly and correctly decoded RM blocks according to their binary error weight. Conditionally on the individual RM block weights $(w_i)_{0 \le i < n_1}$, we model the decoding outcomes by independent Bernoulli variables $F_i \sim \text{Bernoulli}(q_{w_i})$, where $F_i = 1$ indicates that the i-th RM block is incorrectly decoded. The number of erroneous RS symbols is therefore modeled as $w_{\text{RS}} = \sum_{i=0}^{n_1-1} F_i$.

Since q_j is derived from an upper bound, the resulting probabilities are estimates, not exact properties of the implementation.

Conditional error-weight distribution. Applying the generating-function argument of [BDM+26] to the $n_1 n_2$ retained positions gives the following conditional distribution and expectation:

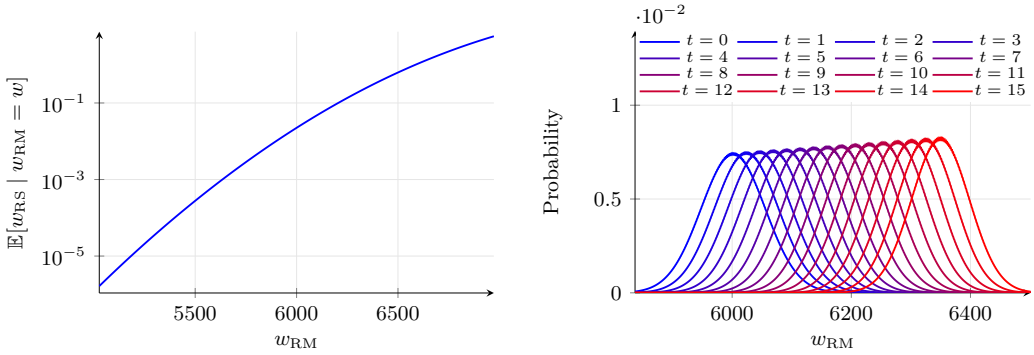
Proposition 3.

$$\Pr(w_{\text{RS}} = t \mid w_{\text{RM}} = w) = \frac{\binom{n_1}{t}[z^w] B(z)^t G(z)^{n_1 - t}}{\binom{n_1 n_2}{w}}. \tag{18}$$

$$\mathbb{E}[w_{\text{RS}} \mid w_{\text{RM}} = w] = \frac{n_1 [z^w] B(z)(1+z)^{n_1 n_2 - n_2}}{\binom{n_1 n_2}{w}}. \tag{19}$$

Table 7: Conditional expectations of the truncated error weight and probabilities of the number of erroneous RS symbols for HQC-1, computed under Model 2 from [BDM+26].

t	E_t	p_t	t	E_t	p_t
0	6001.651	0.975592	8	6183.267	2.98604×10^{-19}
1	6023.894	0.0241317	9	6206.580	4.01652×10^{-22}
2	6046.264	2.74525×10^{-4}	10	6230.034	4.41113×10^{-25}
3	6068.764	1.91214×10^{-6}	11	6253.631	3.98738×10^{-28}
4	6091.396	9.15943×10^{-9}	12	6277.372	2.98512×10^{-31}
5	6114.160	3.21324×10^{-11}	13	6301.260	1.85976×10^{-34}
6	6137.059	8.58500×10^{-14}	14	6325.296	9.67774×10^{-38}
7	6160.094	1.79364×10^{-16}	15	6349.482	4.21808×10^{-41}



(a) $\mathbb{E}[w_{RS} \mid w_{RM} = w]$. (b) w_{RM} conditioned on $w_{RS} = t$.

Figure 8: Relationship between w_{RM} and the number of erroneous RS symbols for HQC-1, computed using [BDM+26, Model 2].

Applying Bayes' theorem to Proposition 3 then gives

$$\Pr(w_{RM} = w \mid w_{RS} = t) = \frac{\pi_w \Pr(w_{RS} = t \mid w_{RM} = w)}{\sum_{u=0}^{n_1 n_2} \pi_u \Pr(w_{RS} = t \mid w_{RM} = u)}. \tag{20}$$

This posterior distribution provides an estimate of the truncated error weight from an observation of w_{RS}. The resulting numerical values are reported in Table 7.

Table 7 shows that the expected full error weight increases with the number of erroneous RS symbols in the considered range. For example, the expected full error weight increases from approximately 6002 when $w_{RS} = 0$ to 6024 when $w_{RS} = 1$. This difference indicates that the number of erroneous RS symbols can provide statistical information about the Hamming weight of the underlying binary error. However, the distribution of w_{RS} is highly concentrated around zero. In particular, the model predicts $\Pr(w_{RS} = 0) \approx 0.976$. Thus, most decapsulations produce no erroneous RS symbols, and observations with several erroneous symbols are particularly rare. Consequently, although the number of RS symbol errors contains information about the error weight, it provides a considerably coarser observation than the individual RM error weights exploited in our CCA and KCA. Figure 8a illustrates the conditional expectation of the number of erroneous RS symbols as a function of the full error weight. Figure 8b further illustrates the posterior distributions of the full error weight for different values of w_{RS}.

Side-channel implications. If an attacker can recover the degree of the error-locator polynomial through side-channel leakage, for example by detecting zero high-degree coefficients in its array representation, they may learn w_{RS} whenever the number of symbol errors

is within the RS correction radius $t_{\mathrm{RS}} = \lfloor \frac{d_{\mathrm{RS}}-1}{2} \rfloor$, assuming that the decoder correctly computes the corresponding locator polynomial. Knowledge of w_{RS} could then provide an indirect estimate of the truncated error weight, using the conditional distribution of Equation 20. Such observations could potentially be exploited in a distance-spectrum recovery attack. However, their usefulness would depend on the amount of information revealed by the RS decoder and the ability to distinguish the different values of w_{RS} from physical leakage. Further investigations are therefore necessary to determine whether leakage from the RS decoding procedure can provide sufficiently accurate error-weight estimates for practical key recovery.

Nevertheless, this analysis highlights that the RS decoder may represent an additional source of sensitive information, motivating the protection of both the RM and RS decoding procedures.

7 Conclusion

In this work, we demonstrate a practical side-channel attack targeting the Reed–Muller decoding of HQC enabling secret key recovery by exploiting leakage from the decryption error weight. We investigate two attack scenarios: a chosen-ciphertext attack on the ExpandAndSum function and a known-ciphertext attack on the FindPeaks function. We propose a new algorithm based on simulated annealing to reconstruct the secret key from a soft spectrum recovered from noisy leakage. Our approach is evaluated through simulations and compared to the reconstruction algorithm from [SSV26], showing a reduction in the number of required error weight observations by a factor of 4.4 to 5.8 for empirical success rates of 50% and 95%, respectively. The average execution time of our algorithm is also reduced compared to previous techniques. While the previous algorithm can reach an average execution time of thousands of seconds, our approach runs on average in about 40 seconds for all evaluated success rates. We validate our technique on an ARM Cortex-M4 platform using real side-channel measurements and successfully recover the secret key with 100 000 traces in the CCA setting and 1 000 000 in the KCA setting with a 100% success rate in both cases. In particular, our algorithm achieves a 93.5% success rate with 40 000 traces in the CCA setting, compared to 200 000 traces for the algorithm from [SSV26], corresponding to a fivefold reduction. More importantly, in the more restrictive KCA setting, our simulated annealing algorithm recovers the secret key in all 200 trials with 1 000 000 traces, whereas the previous reconstruction algorithm fails to recover the key in any of the 200 trials, even with 1 300 000 traces. Finally, we discuss how masking could thwart our attacks at the cost of an overhead in performance.

Future work could focus on improving the efficiency of our reconstruction algorithm by optimizing its parameters or exploring alternative score functions. Moreover, we investigate another potential source of error-weight leakage in the Reed–Solomon (RS) decoder. Using a probabilistic model, we establish a statistical relationship between the number of erroneous RS symbols and the Hamming weight of the full error pattern. A next step would therefore be to investigate whether such leakage can be observed in practice and exploited for secret-key recovery.

These results confirm the importance of preventing leakage of the error weight in HQC implementations as it can ultimately lead to secret key recovery, even though the relationship between the error weight and the secret key is not direct.

Acknowledgement

This research was partially funded by the French Agence Nationale de la Recherche and plan France 2030 program under grant ANR-22-PETQ-0008 PQ-TLS. Gabriel Zaid's research is funded by the European Union (ERC, AMAskZONE, 101077506). Views and opinions expressed are however those of the author only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

A Signal-to-Noise Ratios

Through this section, we illustrate the implementation analysis conducted in Section 3 by computing the *Signal-to-Noise Ratio* on the Hamming weight of $\mathbf{e}'$ when ExpandAndSum and FindPeaks are respectively executed.

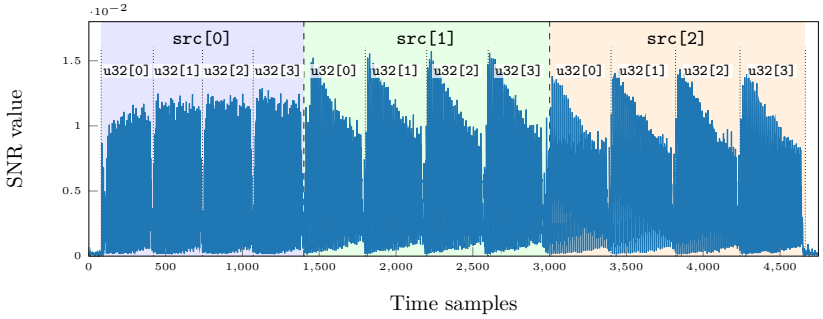


Figure 9: SNR of ExpandAndSum function with respect to the Hamming weight of $\mathbf{e}'$ (over 384 bits) for 10 000 decapsulation traces (*i.e.* 460 000 sub-traces) with $\mathbf{m} = \mathbf{0}$.

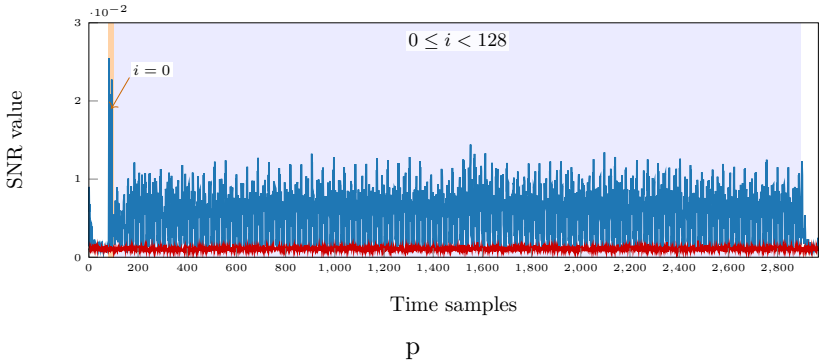


Figure 10: SNR of the FindPeaks function with respect to the Hamming weight of $\mathbf{e}'$ (over 384 bits) for 70 000 sub-traces with $\mathbf{m} = \mathbf{0}$ (blue curve), compared with the SNR obtained using random labels (red curve). The first iteration, highlighted in orange, corresponds to the processing of the correct Hadamard coefficient when $\mathbf{m} = \mathbf{0}$.

References

[BDM+26] Sebastian Bitzer, Jean-Christophe Deneuville, Emma Munisamy, Bharath Purtipli, Stefan Ritterhoff, and Antonia Wachter-Zeh. HQC beyond the standard: Ciphertext compression and refined DFR analysis. In Joan Daemen

and Emmanuel Thomé, editors, *Advances in Cryptology - EUROCRYPT 2026 - 45th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Rome, Italy, May 10-14, 2026, Proceedings, Part VI*, volume 16546 of *Lecture Notes in Computer Science*, pages 544–574. Springer, 2026.

[BMG+25] Chloé Baïsse, Antoine Moran, Guillaume Goy, Julien Maillard, Nicolas Aragon, Philippe Gaborit, Maxime Lecomte, and Antoine Loiseau. Secret and shared keys recovery on hamming quasi-cyclic with SASCA. *Des. Codes Cryptogr.*, 93(6):2137–2157, 2025.

[CK13] Omar Choudary and Markus G. Kuhn. Efficient template attacks. In Aurélien Francillon and Pankaj Rohatgi, editors, *Smart Card Research and Advanced Applications*, pages 253–270, Cham, 2013. Springer International Publishing.

[CRR03] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski, Jr., Çetin Kaya Koç, and Christof Paar, editors, *CHES 2002*, volume 2523 of *LNCS*, pages 13–28. Springer, Berlin, Heidelberg, August 2003.

[CS20] Gaëtan Cassiers and François-Xavier Standaert. Trivially and efficiently composing masked gadgets with probe isolating non-interference. *IEEE Transactions on Information Forensics and Security*, 15:2542–2555, 2020.

[DG25a] Haiyue Dong and Qian Guo. Multi-value plaintext-checking and full-decryption oracle-based attacks on HQC from offline templates. *IACR TCHES*, 2025(4):254–289, 2025.

[DG25b] Haiyue Dong and Qian Guo. OT-PCA: New key-recovery plaintext-checking oracle based side-channel attacks on HQC with offline templates. *IACR TCHES*, 2025(1):251–274, 2025.

[DG26] Haiyue Dong and Qian Guo. Breaking optimized HQC: the first cache-timing full decryption oracle key-recovery attack in post-quantum cryptography. In Nadia Heninger and Mike Rosulek, editors, *Advances in Cryptology - CRYPTO 2026 - 46th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-20, 2026, Proceedings, Part IV*, volume 16803 of *Lecture Notes in Computer Science*, pages 224–255. Springer, 2026.

[DGN26] Haiyue Dong, Qian Guo, and Denis Nabokov. Single-trace key recovery attacks on HQC using valid and invalid ciphertexts. In Joan Daemen and Emmanuel Thomé, editors, *EUROCRYPT 2026, Part VII*, volume 16547 of *LNCS*, pages 568–597. Springer, Cham, May 2026.

[ELPS18] Edward Eaton, Matthieu Lequesne, Alex Parent, and Nicolas Sendrier. QC-MDPC: A timing attack and a CCA2 KEM. In Tanja Lange and Rainer Steinwandt, editors, *PQCrypto 2018*, pages 47–76. Springer, Cham, 2018.

[FHS+17] Tomás Fabsic, Viliam Hromada, Paul Stankovski, Pavol Zajac, Qian Guo, and Thomas Johansson. A reaction attack on the QC-LDPC McEliece cryptosystem. In Tanja Lange and Tsuyoshi Takagi, editors, *PQCrypto 2017*, pages 51–68. Springer, Cham, 2017.

[GAMA+25] Philippe Gaborit, Carlos Aguilar-Melchor, Nicolas Aragon, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Edoardo Persichetti, Gilles Zémor, Jurjen Bos, Arnaud Dion, Jérôme Lacan, Jean-Marc Robert, Pascal Véron, Paulo L. Barreto, Santosh Ghosh, Shay Gueron, Tim Güneysu,

Rafael Misoczki, Jan Richter-Brokmann, Nicolas Sendrier, Jean-Pierre Tillich, and Valentin Vasseur. Hamming quasi-cyclic (HQC), August 2025. HQC specifications, version 22 August 2025.

[GHL+26] Yikang Guo, Yuchang Hu, Yanbin Li, Shilin Sun, Geng Chen, Chunpeng Ge, and Jiankuo Dong. From multiplication to recovery: A single-trace side-channel attack on hqc. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2026(4):890–916, Sep. 2026.

[GJS16] Qian Guo, Thomas Johansson, and Paul Stankovski. A key recovery attack on MDPC with CCA security using decoding errors. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 789–815. Springer, Berlin, Heidelberg, December 2016.

[GLG22] Guillaume Goy, Antoine Loiseau, and Philippe Gaborit. A new key recovery side-channel attack on HQC with chosen ciphertext. In Jung Hee Cheon and Thomas Johansson, editors, *PQCrypto 2022*, pages 353–371. Springer, Cham, September 2022.

[GSA+25] Guillaume Goy, Maxime Spyropoulos, Nicolas Aragon, Philippe Gaborit, Renaud Pacalet, Fabrice Perion, Laurent Sauvage, and David Vigilant. Side-channel sensitivity analysis on HQC: Towards a fully masked implementation. Cryptology ePrint Archive, Paper 2025/1344, 2025.

[HHK17] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the fujisaki-okamoto transformation. In Yael Kalai and Leonid Reyzin, editors, *Theory of Cryptography*, pages 341–371, Cham, 2017. Springer International Publishing.

[HQC] Hqc gitlab repository. https://gitlab.com/pqc-hqc/hqc.

[KGV83] Scott Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 05 1983.

[MNMD25] Nathan Maillet, Cyrius Nugier, Vincent Migliore, and Jean-Christophe Deneuville. Key recovery from side-channel power analysis attacks on non-SIMD HQC decryption. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part V*, volume 16004 of *LNCS*, pages 70–102. Springer, Cham, August 2025.

[MOP07] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks – Revealing the Secrets of Smart Cards*. Springer US, 2007.

[Nik17] Ivica Nikolic. How to use metaheuristics for design of symmetric-key primitives. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part III*, volume 10626 of *LNCS*, pages 369–391. Springer, Cham, December 2017.

[OII+25] Motonari Ohtsuka, Takahiro Ishimaru, Rei Iseki, Shingo Kukita, and Kohtaro Watanabe. Key reconstruction for QC-MDPC McEliece from imperfect distance spectrum. Cryptology ePrint Archive, Report 2025/485, 2025.

[OIT+25] Motonari Ohtsuka, Takahiro Ishimaru, Yuta Tsukie, Shingo Kukita, and Kohtaro Watanabe. Efficient reconstruction in key recovery attack on the QC-MDPC mceliece cryptosystems. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, 108(3):414–422, 2025.

[PRJ+25] Thales B. Paiva, Prasanna Ravi, Dirmanto Jap, Shivam Bhasin, Sayan Das, and Anupam Chattopadhyay. Et tu, brute? Side-channel assisted chosen ciphertext attacks using valid ciphertexts on HQC KEM. In Ruben Niederhagen and Markku-Juhani O. Saarinen, editors, *Post-Quantum Cryptography - 16th International Workshop, PQCrypto 2025, Part II*, pages 294–321. Springer, Cham, April 2025.

[PT18] Thales Bandiera Paiva and Routo Terada. Improving the efficiency of a reaction attack on the QC-MDPC mceliece. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, 101-A(10):1676–1686, 2018.

[PT19] Thales Bandiera Paiva and Routo Terada. A timing attack on the HQC encryption scheme. In Kenneth G. Paterson and Douglas Stebila, editors, *SAC 2019*, volume 11959 of *LNCS*, pages 551–573. Springer, Cham, August 2019.

[SSV26] Magali Salom, Nicolas Sendrier, and Valentin Vasseur. Sparse vector reconstruction from distance spectrum using soft information. In Magali Bardet and Ruben Niederhagen, editors, *Post-Quantum Cryptography - 17th International Workshop, PQCrypto 2026, Saint-Malo, France, April 14-16, 2026, Proceedings, Part I*, volume 16491 of *Lecture Notes in Computer Science*, pages 99–129. Springer, 2026.

[YBL26] Luhan Yan, Zhenzhen Bao, and Huina Li. Refining probabilistic-linearization TIDA for 5-round SHA3-384 collision attacks (full version). Cryptology ePrint Archive, Paper 2026/2090, 2026.