Secret and Shared Keys Recovery on Hamming Quasi-Cyclic with SASCA

Chloé Baïsse[1,2], Antoine Moran[2,3,4], Guillaume Goy[1,2], Julien Maillard[1,2], Nicolas Aragon[1], Philippe Gaborit[1], Maxime Lecomte[2], and Antoine Loiseau[2]

[1] XLIM - University of Limoges, Limoges, France
{chloe.baisse,guillaume.goy,nicolas.aragon,gaborit}@unilim.fr
[2] Univ. Grenoble Alpes, CEA, Leti. MINATEC Campus, F-38054 Grenoble, France
{antoine.moran,julien.maillard,maxime.lecomte,antoine.loiseau}@cea.fr
[3] Inria, 91120 Palaiseau, France
[4] Polytechnic Institute of Paris, 91120 Palaiseau, France.

Abstract. Soft Analytical Side Channel Attacks (SASCA) are a powerful family of Side Channel Attacks (SCA) that allow to recover secret values with only a small number of traces. Their effectiveness lies in the Belief Propagation (BP) algorithm, which enables efficient computation of the marginal distributions of intermediate values. Post-quantum schemes such as Kyber, and more recently, Hamming Quasi-Cyclic (HQC), have been targets of SASCA. Previous SASCA on HQC focused on Reed-Solomon (RS) codes and successfully retrieved the shared key with a high success rate for high noise levels using a single trace. In this work, we present new SASCA on HQC where both the shared key and the secret key are targeted. Unlike the previous SASCA, we take a closer look at the Reed-Muller (RM) code. The advantage of this choice, is that the RM decoder is applied before the RS decoder. This is what makes it possible to attack the two keys. We build a factor graph of the Fast Hadamard Transform (FHT) function from the HQC reference implementation of April 2023. The information recovered from BP allows us to retrieve the shared key with a single trace. In addition to the previous SASCA targeting HQC, we also manage to recover the secret key with two chosen ciphertext attacks. One of them require a single trace and is successful until high noise levels.

Keywords: Soft Analytical Side Channel Attacks (SASCA) · Belief Propagation (BP) · Hamming Quasi-Cyclic (HQC) · Reed-Muller (RM) Codes · Post-Quantum Cryptography (PQC) · Single Trace Attacks · Chosen Ciphertexts Attacks (CCA)

Introduction

Hamming Quasi-Cyclic (HQC) [19] is a code-based Key Encapsulation Mechanism (KEM), and a candidate for the National Institute of Standards and Technology (NIST)'s contest for the standardisation of post-quantum cryptosystems. After three rounds, the NIST selected the KEM CRYSTALS-Kyber [6] as well as the signature schemes CRYSTALS-Dilithium [7], FALCON [24] and SPHINCS+ [5]. The first three are lattice-based, and SPHINCS+ is hash-based. A fourth round is currently taking place between the three code-based KEM Classic McEliece [2], BIKE [3] and HQC. During the course of the contest, the security of the candidates have been tested against Side-Channel Attacks (SCA),

and post-quantum schemes were found to be vulnerable to this kind of attacks. Indeed, as SCA rely on a physical access to the device on which the scheme is implemented, the security of the implementation holds more importance than the hardness of the problem the scheme is based on. Protection against SCA is one of the metric used to evaluate candidates in the NIST's process [1].

Concerning HQC, a first version of the cryptosystem that uses BCH codes has been targeted by two Timing Attacks (TA) in 2019 [31] and 2020 [21], as well as a chosen ciphertext attack [29] in 2020. During the third round of the NIST contest, authors of HQC made the previous attacks out-of-date with a new version of the cryptosystem based on concatenated Reed-Muller (RM) and Reed-Solomon (RS) codes.

A message recovery attack with a single electromagnetic trace by Goy et al. [9], exploited the error free RS codeword decoding due to the small Decryption Failure Rate. However, the attack needed more than 2^{96} algebraic operations to recover the shared key from the side channel information, which is not realistic in practice.

In 2022, the RM decoder has been targeted by two chosen ciphertext attacks where the support of the secret key $\boldsymbol{y}$ is recovered by the use of an oracle. In the attack of Schamberger et al. [28], the oracle determined if the decoded word is the all-zero codeword or not, while in the attack of Goy et al. [8], the oracle gave the number of errors corrected by the RM decoder.

Soft Analytical Side Channel Attacks (SASCA) were first proposed in 2014 by Veyrat-Charvillon et al. on the cryptosystem AES [30]. The idea is to combine the side-channel information of the intermediate variables with the Belief Propagation (BP) algorithm [16], in order to find the full key. The result is a powerful, noise-resistant generic attack with low timing and memory complexities. With SASCA, authors were able to attack the AES Furious implementation with a single trace for small noises, and with less traces than a standard template attack for larger noises.

In 2020, Kannwischer et al. tried SASCA for different kind of scenarios [15], on the Keccak hash function that often deals with ephemeral secrets. Their results show that single trace attacks are a serious threat against Keccak.

Related Work The lattice-based cryptosystem Kyber [6], standardised by NIST, has also been a target of SASCA with five attacks [25,22,12,13,4]. All of them focused on the Number Theoretic Transform (NTT) or its inverse, used in the scheme to perform polynomial multiplications efficiently.

These operations have a structure that makes heavy use of load and store operations, which are known to produce a high amount of side-channel leakage. There are composed of layers where an operation called a butterfly, is performed on two coefficients of a polynomial to compute two coefficients of a new one.

The first attack was proposed in 2017 by Primas et al. [25] and used a single trace attack. Authors performed simulations with a Hamming weight leakage model and obtained a success rate superior to 0.9, up to a σ of 0.4 for both the masked and unmasked cases. In 2019, Pessl et al. [22] improved the former attack by reducing number of templates from one million to 213 Hamming weight templates. They also used some strategies to help the convergence of the BP algorithm such as scheduling, damping and reducing cycles. On the simulated case, their success rate is good up to a σ of 1.5 for the unmasked case and 0.3 for the masked one. In 2021, Hamburg et al. applied a k-trace SASCA ($k \in \{2, 3, 4\}$) using a

chosen ciphertext strategy [12], in order to have a sparse input linked to the secret. They obtained a good success rate up to $\sigma \leqslant 1,2$ for k traces and up to $\sigma \leqslant 0.5-0.7$ for a single trace. Countermeasures against single trace attacks on the NTT have been propound in 2020 by Ravi et al. [27], including Fine Shuffling and Coarse Shuffling. In 2023, these two shuffling countermeasures have been tested by Hermelink et al. [13] against the attack of Hamburg et al. of 2021 [12]. While they were found to be efficient, some tools have been introduced to weaken them. Assael et al. attacked an optimized implementation of the Kyber for ARM Cortex M4 in 2023 [4]. They introduced message pruning, a new technique to improve the run-time of BP. Their attack has a high success rate until a σ of 1.2.

In 2023, Goy et al. led the first SASCA on HQC [10], targeting the shared key manipulated by the Reed Solomon (RS) code. Their simulated attack with a Hamming weight leakage model is able to reach a high success rate up to a σ of 2 or 3, depending on the security level . Against a masking countermeasure, the success rate is still good under a σ of 1, which does not provide security. They also adapted the shuffling countermeasures used to protect Kyber in [27] and showed that they were not efficient. They introduced full shuffling that mixed the previously tested countermeasures, and proved that the added combinatorial complexity is enough to prevent the attack. Finally, they attacked the decapsulation of HQC by combining the intermediate variables' information of the RS decoder and the RS encoder in a same instance of BP. This strategy allowed them to increase the noise level up to which the attack works.

Our Contributions In this paper, we present two physical side-channel attacks against HQC based on Belief Propagation. These attacks target the HQC decoder, specifically focusing on the Fast Hadamard Transform (FHT) involved in the Reed-Muller decoding algorithm. Since the FHT structure is similar to the NTT, our work is inspired by the attacks on the NTT using SASCA. We provide a factor graph of this operation and devise two distinct attacks by exploiting leaks in each of the intermediate operations of the FHT. The attacks are performed on simulations produced with a Hamming weight leakage model.

- We demonstrate that during a normal execution of HQC, the leaks observed from the computation of the Fast Hadamard Transform (FHT) are sufficient to recover the HQC shared key. This attack can benefit from the re-decoding technique, introduced by Goy et al. [9,10], to correct errors that appear during the attack, thereby improving the accuracy of the attack.
- We present two chosen-ciphertext attacks that allow the recovery of the HQC secret key by exploiting the same leaks as in the previous attack.
  - By using the Information Set Decoding (ISD) algorithm, we are able to conduct a successful attack on the secret key with one chosen ciphertext, and thus, with a single trace attack.
  - Without ISD, we show that, depending on HQC security level, 3 or 5 judiciously chosen ciphertexts are enough to recover the HQC secret key.

We simulate these three attacks assuming an Hamming weight leakage model and we show that we are able to recover HQC secrets values even with high noise levels. Finally, we propose countermeasures.

1 Background

1.1 Hamming Quasi-Cyclic

General scheme Hamming Quasi-Cyclic (HQC) [19] is a code-based Key Encapsulation Mechanism (KEM) that allows to share a session key in order to communicate with symmetric cryptography. Its security is assured by a reduction of the quasi-cyclic syndrome decoding problem, established by a quasi-cyclic code of parameters $[2n, n]$. The scheme uses a linear code $\mathcal{C}$ of parameters $[n, k]$ of generator matrix $\mathbf{G} \in \mathbb{F}_2^{k \times n}$, which decoding algorithm is publicly known. The KEM version of HQC (HQC-KEM) has been built by applying the transformation of Hofheinz-Hövelmanns-Kiltz (HHK) [14] on the Public Key Encryption version of HQC (HQC-PKE).

HQC-PKE has been proven to have an IND-CPA security, and the application of the HHK transformation allows HQC-KEM to reach an IND-CCA2 security. HQC-KEM embodies the following algorithms of Key Generation, Encryption and Decryption of HQC-PKE, where $\mathcal{R}$ denotes the polynomial ring $\mathbb{F}_2[X]/(X^n - 1)$, and $\mathcal{R}_\omega = \{\boldsymbol{x} \in \mathcal{R} \ ; \ \mathtt{HW}(\boldsymbol{x}) = \omega\}$.

Algorithm 1 KeyGen

Input: parameters
$\boldsymbol{h} \xleftarrow{\$} \mathcal{R}$
$(\boldsymbol{x}, \boldsymbol{y}) \xleftarrow{\$} \mathcal{R}_\omega^2$
$s_k \leftarrow (\boldsymbol{x}, \boldsymbol{y})$
$\boldsymbol{s} \leftarrow \boldsymbol{x} + \boldsymbol{hy}$
$p_k \leftarrow (\boldsymbol{h}, \boldsymbol{s})$
return (s_k, p_k)

Algorithm 2 Encrypt

Input: p_k, $\boldsymbol{m}$
$\boldsymbol{e} \xleftarrow{\$} \mathcal{R}_{\omega_e}$
$(\boldsymbol{r}_1, \boldsymbol{r}_2) \xleftarrow{\$} \mathcal{R}_{\omega_r}^2$
$\boldsymbol{u} \leftarrow \boldsymbol{r}_1 + \boldsymbol{hr}_2$
$\boldsymbol{v} \leftarrow \boldsymbol{m}\mathbf{G} + \boldsymbol{sr}_2 + \boldsymbol{e}$
$\boldsymbol{c} \leftarrow (\boldsymbol{u}, \boldsymbol{v})$
return c

Algorithm 3 Decrypt

Input: s_k, $\boldsymbol{c}$
return $\mathcal{C}$.Decode$(\boldsymbol{v} - \boldsymbol{uy})$

In HQC-KEM, the encryption function is de-randomised thanks to a seed depending on the message (a hashed message). This allows to re-encrypt the message after the decryption in order to verify if the received ciphertext is equal to the newly computed one. The session key, that corresponds to a hash of the concatenation of the message and the ciphertext, can be shared only if the equality is verified. The KEM is produced by Algorithm 4 and Algorithm 5, where $\mathcal{G}, \mathcal{H}$ and $\mathcal{K}$ are hash functions.

Algorithm 4 Encapsulate

Input: p_k
$\boldsymbol{m} \xleftarrow{\$} \mathbb{F}_2^k$
$\theta \leftarrow \mathcal{G}(\boldsymbol{m})$
$\mathbf{c} \leftarrow$ Encrypt$(p_k, \boldsymbol{m}, \theta)$
$K \leftarrow \mathcal{K}(\boldsymbol{m}, \boldsymbol{c})$ ▷ shared key
$\boldsymbol{d} \leftarrow \mathcal{H}(\boldsymbol{m})$
return $(\boldsymbol{c}, \boldsymbol{d})$

Algorithm 5 Decapsulate

Input: $s_k, \boldsymbol{c}, \boldsymbol{d}$
$\boldsymbol{m} \leftarrow$ Decrypt$(s_k, \boldsymbol{c})$
$\theta' \leftarrow \mathcal{G}(\boldsymbol{m}')$
$\boldsymbol{c}' \leftarrow$ Encrypt$(p_k, \boldsymbol{m}', \theta')$
if $\boldsymbol{c} \neq \boldsymbol{c}'$ or $\boldsymbol{d} \neq \mathcal{H}(\boldsymbol{m}')$ **then**
  return $\perp$
else
  return $\mathcal{K}(\boldsymbol{m}, \boldsymbol{c})$
end if

The code $\mathcal{C}$ used in HQC is a concatenated code with a duplicated Reed-Muller (RM) code over $\mathbb{F}_2$ as an internal code, and a shortened Reed-Solomon (RS) code over $\mathbb{F}_{2^8}$ as

an external code. Let us denote by $[n_1, k_1]$ the parameters of the shortened RS code, and by $[n_2, k_2]$ the parameters of the duplicated RM code. Given the construction of a concatenated code, the code $\mathcal{C}$ would have parameters $[n_1 n_2, k_1 k_2]$. However, in order to thwart structural attacks, the parameters of $\mathcal{C}$ are $[n, k_1 k_2]$ where n is the smallest primitive prime superior to $n_1 n_2$.

Security Level	`mul`	k_1	k_2	n_1	n_2	$n_1 n_2$	n	l	ω	$\omega_e = \omega_r$
HQC-128	3	16	8	46	384	17664	17669	5	66	75
HQC-192	5	24	8	56	640	35840	35851	11	100	114
HQC-256	5	32	8	90	640	57600	57637	37	131	149

Table 1. HQC parameters from [19]

Reed-Muller codes The duplicated RM code of HQC uses a $RM(1, 7)$ code of parameters $[128, 8, 64]$. The encoding is first performed in the same way as for a classical $RM(1, 7)$ code. Then, the codeword is duplicated a given number of times, defined by the multiplicity parameter `mul` of HQC. HQC-128 has a multiplicity of 3, which changes the parameters of the code to $[384, 8, 192]$. In HQC-192 and HQC-256, the multiplicity is equal to 5 and the duplicated RM code has parameters $[640, 8, 320]$. The duplicated RM code is applied independently to blocks of length k_2 during the encoding step, and of length n_2 during the decoding step. The decoding algorithm of the duplicated RM code is done in three steps (see Figure 1)

The main step performed during the RM decoding is the Fast Hadamard Transform (FHT). This decoding method can be used for $RM(1, m)$ codes, as they can be seen as Hadamard codes, which has been proved by MacWilliams and Sloane in [18]. In the case of HQC, the FHT takes as input a RM codeword of length 128 called expanded codeword and return a transformed codeword of the same length. In the reference implementation [19] of April 2023, the FHT is implemented following Algorithm 6.

Algorithm 6 Fast Hadamard Transform from [19]

```
void hadamard (expandedCodeword *src, expandedCodeword *dst) {
    expandedCodeword *p1 = src;
    expandedCodeword *p2 = dst;
    for (int32_t pass = 0; pass < 7; pass++) {
        for (int32_t i = 0; i < 64; i++) {
            (*p2)[i] = (*p1)[2 * i] + (*p1)[2 * i + 1];
            (*p2)[i + 64] = (*p1)[2 * i] - (*p1)[2 * i + 1];
        }
        expandedCodeword *p3 = p1;  // swap p1, p2 for next round
        p1 = p2;
        p2 = p3;
    }
}
```

Decoding $\boldsymbol{v} - \boldsymbol{uy}$ In HQC, the vector $\boldsymbol{z} = \boldsymbol{v} - \boldsymbol{uy}$ to be decoded is not a regular RM codeword but a concatenation of n_1 duplicated RM codewords of length n_2. The first steps of the algorithm is then to isolate the n_1 chunks and remove their multiplicity, just before applying the FHT.

The value $\boldsymbol{z} = \boldsymbol{v} - \boldsymbol{uy}$ of length n is divided into n_1 chunks of length n_2. The remaining $l = n - n_1 n_2$ bits are truncated, and are not used for the RM decoding process. Each of these n_1 chunks is manipulated independently by a RM decoder. We focus on the description of one of these decoders, summarized in Figure 1.

The first step of the decoder is to apply the expand and sum function This algorithm takes as input a chunk of length $n_2 = 128 \times \texttt{multiplicity}$, namely $\boldsymbol{z}_i = \left(\boldsymbol{z}_i^{(1)} \mid \boldsymbol{z}_i^{(2)} \mid \cdots \mid \boldsymbol{z}_i^{(\texttt{mul})}\right)$ and returns as output $\dot{\boldsymbol{z}}_i = \sum_{k=1}^{\texttt{mul}} \boldsymbol{z}_i^{(k)}$, the sum of each bloc (this operation is the natural sum over $\mathbb{N}$). The expand and sum algorithm then returns a vector $\dot{\boldsymbol{z}}_i$ of length 128 whose each value leaves in $[0, \texttt{mul}]$. This vector is the input of the FHT, which outputs a vector $\boldsymbol{w}_i \in [-64 \cdot \texttt{mul}, 128 \cdot \texttt{mul}]^{128}$. After a slight correction on the first coordinate of this vector, the find peaks algorithm is applied. Find peaks aims at locating the maximum coordinates of the vector under absolute value. The location of the maximum value as well as its sign, are encoded into 8 bits of information denoted as $\boldsymbol{c}_i$, which is the final output of the RM decoder.

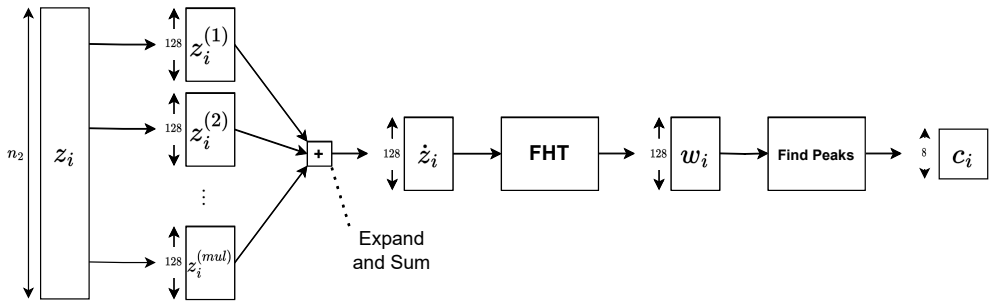

Fig. 1. Structure of the HQC expanded Reed-Muller decoder for one chunk $\boldsymbol{z}_i$.

Reed-Solomon decoding After the RM decoding step, the n_1 outputs $(c_1, \cdots, c_{n_1})$ represent an erroneous Reed-Solomon codeword that is decoded by the RS decoder. In this paper, we do not describe the RS decoding process, as we do not target this operation during our attack. However, the RS decoder can be useful to take advantage of the re-decoding strategy, introduced by Goy et al. [9,10]. This strategy aims at correcting the errors made during the attacks exploiting the knowledge of the codeword structure. The method benefits from the low Decryption Failure Rate (DFR) of the duplicated RM decoder of HQC (see Table 2), ensuring that the message as a high probability of being decoded before the RS decoder is applied. For the HQC parameters, we give the error

correction capabilities of the classical decoder and the Guruswami-Sudan list decoder [11] in Table 2.

Security level	HQC parameters			List decoder	RM observed
λ	k_1	n_1	t	τ_{GS}	DFR
HQC-128	16	46	15	19	$2^{-10.96}$
HQC-192	24	56	16	19	$2^{-14.39}$
HQC-256	32	90	29	36	$2^{-11.48}$

Table 2. Reed-Solomon error correction capability of the RS decoder for each HQC set of parameters, given for a classical decoder and the Guruswami-Sudan list decoder, as well as the duplicated RM decoder observed DFR.

1.2 Soft Analytical Side Channel Attacks

Belief Propagation (BP) is an efficient message-passing algorithm commonly used in the fields of artifical intelligence or information theory, to compute inferences or marginal probabilities on a graphical model. Some examples of application are Bayesian Networks, Markov random fields, and the decoding of Low Density Parity Check codes.

During a SCA, an attacker generally aims at obtaining a probability distribution of intermediate values of a cryptographic function. Soft Analytical Side Channel Attacks (SASCA) go further by connecting the gathered side-channel information in a graphical model linking the intermediate values according to the performed mathematical operations. By appliying BP on this model, the attacker is able to effectively compute the marginal distributions of the intermediate variables, which will lead to finding the marginal distribution of the secret.

Building the factor graph The graphical model that links the intermediate values accordingly to the computation steps of the algorithm is called a factor graph. It is a bipartite graph with variable nodes and factor nodes.

A variable node represents an intermediate value. A factor node can embody an operation of the algorithm. The goal is to check the consistency of the operation for the possible values of the intermediate variables taking part in the operation. The nodes of those intermediate variables are linked by an edge to the factor node. There are also observational factor nodes, that hold the probability distribution of a variable found by a template attack. In that case, an edge exists between the variable node and the observational factor node.

Applying the Belief Propagation algorithm The Belief Propagation (BP) algorithm [16] aims at computing the marginals of every intermediate variable of the attacked function. Thanks to a message-passing principle, the information in the nodes of the factor graph is propagated and updated during each iteration of BP.

The following formulas as well as a clear description of BP can be found in the chapter 26 of [17]. Let $\mathbf{x} = \{x_n, 1 \leqslant n \leqslant N\}$ be a set of N variables. Let $\{f_m, 1 \leqslant m \leqslant M\}$ be a set of M factors, where each f_m take as argument a subset $\mathbf{x}_m$ of $\mathbf{x}$. $\mathcal{N}(m)$ refers to the set of indexes of the neighboring variables of the factor f_m. $\mathcal{M}(n)$ denotes the set of indexes of the neighboring factors of the variable x_n.

Two types of message are passed accordingly to the edges of the factor graph.

A message from the variable x_n to the factor f_m is defined by:

$$q_{n\to m}(x_n) = \prod_{m' \in \mathcal{M}(n)\setminus\{m\}} r_{m'\to n}(x_n) \tag{1}$$

The second type of message is sent by a factor f_m to a variable x_n and is given by:

$$r_{m\to n}(x_n) = \sum_{\mathbf{x}_m \setminus \{x_n\}} \left(f_m(\mathbf{x}_m) \prod_{n' \in \mathcal{N}(m)\setminus\{n\}} q_{n'\to m}(x_{n'}) \right) \tag{2}$$

The marginal distribution of the variable x_n is computed by multiplying all the incoming messages at the node and normalizing.

If BP is applied on a tree-like factor graph, it is proven that convergence is reached and that the found marginals are exact.

Loopy BP Crytptographic operations are often modelled by cyclic factor graphs. In that case, we use an iterative algorithm called loopy BP. The marginals are not necessary correct and the convergence may never happen. In [10], Goy et al. proposed two ways of stopping the loopy BP. First, they define a maximum number of iterations in case the convergence is not reached. Second, they specify a threshold on the maximal statistical change of the variables' distributions to detect convergence. The results of the loopy BP often happened to be good enough for most applications, including the computation of marginal distributions for side channel attacks.

2 Attacking HQC with SASCA

In this section, we detail the generic BP attack strategy employed against HQC. Our approach leads to build attacks on both the shared key and the secret key, with certain elements being common across all attack scenarios.

2.1 Attacker model

In each attack scenario presented in this work, it is assumed that the attacker has access to and full control over a duplicate of the target device. This enables the attacker to execute the profiling phase of a template attack. It is postulated that the attacker is capable of isolating the measurements of the FHT layers as well as those of the butterflies. Although this particular step was not explicitly addressed in our paper, we consider that it is feasible due to the repetition of building blocks in the targeted function. In practice, isolating and ordering operations can be achieved using various techniques, including pattern matching strategies.

In our paper, we suggest that an attacker has the capability to craft templates for each intermediate value of the FHT. Our attack strategy is based on simulated traces, following an Hamming weight leakage model. Numerous preceding studies [10,33] indicate that translating the results of a simulated attack into a practical attack on a smaller target, such as the STM32F407, is not a major challenge. The observed SNR in such targets is low enough to make the attack feasible in real-world scenarios. Nonetheless, executing a practical attack requires both time and abilites to overcome engineering obstacles, which is not a straightforward step. However, given that the attack is comprehensively described and understood theoretically, and the necessary tools for exploiting leaks, like BP on factor graphs, are available, the attack proves to be effective.

2.2 Representing the FHT with a factor graph

Following Algorithm 6, the FHT is structured in 7 layers. Each takes a vector x of 128 integers as input and applies the following operation to compute a new vector y of 128 integers:

$$\forall\ 0 \leqslant i \leqslant 63,\ \begin{cases} y_i &= x_{2i} + x_{2i+1} \\ y_{i+64} &= x_{2i} - x_{2i+1} \end{cases} \tag{3}$$

We call the realization of this operation for one value of i a butterfly. The construction of the FHT factor graph is straightforward and consists of 7 layers of 64 butterflies factor graphs. By directly turning a butterfly into a factor graph, we obtain the factor graph of Figure 2 , where f_a and f_b respectively contain the probability distribution of the variables a and b, and:

$$f_+ = \begin{cases} 1 \text{ if } a+b=c \\ 0 \quad \text{otherwise} \end{cases} \qquad\qquad f_- = \begin{cases} 1 \text{ if } a-b=d \\ 0 \quad \text{otherwise} \end{cases}$$

This butterfly forms numerous and short cycles in the complete FHT factor graph. We choose to apply the clustering strategy [22,12] to eliminate the loops inside each butterfly. In our case, the factor nodes f_+ and f_- are combined in a single factor node f_{bf} such as:

$$f_{bf} = \begin{cases} 1 \text{ if } a+b=c \text{ and } a-b=d \\ 0 \qquad\qquad \text{otherwise} \end{cases} \tag{4}$$

It results in the butterfly of Figure 3. Even if the graph still contains loops due to the FHT structure, these loops are longer, which will leads to increased performance [22].

2.3 Leakage simulations

To realize our simulations, we rely on the Hamming weight leakage model defining the measurement $tr(s)$ of an intermediate variable s by

$$tr(s) = \alpha\, \mathtt{HW}(s) + \mathcal{N}(\beta, \sigma^2).$$

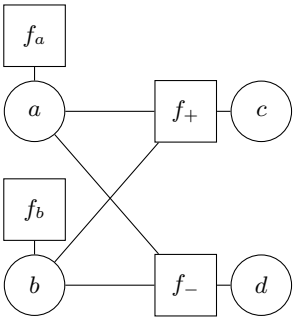

Fig. 2. Direct butterfly FG

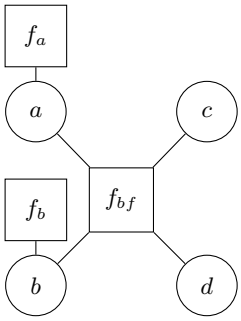

Fig. 3. Chosen butterfly FG

Following this leakage model, we simulate the value r of a linear classifier prediction when the value s is manipulated with the following equation

$$P(r \mid s) = P(X = r)$$

where $X \sim \mathcal{N}(\mathtt{HW}(s), \sigma^2)$.

2.4 Targeting the Fast Hadamard Transform

By recovering the intermediate values of the FHT, an adversary can lead two distinct attacks: one to retrieve the shared session key, and another to retrieve the secret key $\boldsymbol{y}$. The FHT of HQC is thus a highly interesting target for SASCA. In the two following sections, we show how an attacker can retrieve these two secrets thanks to SASCA.

3 Recovering the shared key

3.1 Attacker model

We suppose that the attacker follows the model described in Subsection 2.1. In addition, we consider that an attacker is unable to enhance the Signal-to-Noise Ratio (SNR) using techniques that involve side-channel measurements from multiple instances of HQC decapsulation. Furthermore, since our attack aims at recovering the shared key, which is an ephemeral secret, we only consider a single trace attack scenario. Moreover, the shared key information is contained into the ciphertext value: this implies that we cannot perform a chosen ciphertext attack. The behavior of the targeted FHT is therefore exactly what we observe in a random instance of HQC. We are able to compute the prior knowledge on each intermediate value knowing the probability distribution of the FHT input.

We then build the attack based on the FHT, which allows us to recover the output of the FHT. After applying the remaining RM operations, we are able to recover the shared secret key.

3.2 Prior Knowledge

When the probability distribution of the input vector is known, the structure of the FHT enables to compute analytically the probability distribution of the intermediate values. This information can be integrated into the factor graph as priors inside observational factor nodes.

The value distribution of the input vector The duplicated RM decoder takes $\boldsymbol{v}-\boldsymbol{uy}$ as input. It is a codeword of $\mathcal{C}$ with an error **e'**. Indeed, $\boldsymbol{v}-\boldsymbol{uy}$ is equal to $\boldsymbol{m}\mathbf{G}+\boldsymbol{x}\boldsymbol{r}_2+\boldsymbol{y}\boldsymbol{r}_1+\mathbf{e}$ that we rewrite $\boldsymbol{m}\mathbf{G}+\boldsymbol{e}'$ with $\boldsymbol{e}'=\boldsymbol{x}\boldsymbol{r}_2+\boldsymbol{y}\boldsymbol{r}_1+\boldsymbol{e}$. The input of the FHT can be denoted as $\dot{\boldsymbol{z}}=\mathit{expand\_and\_sum}(\boldsymbol{m}\mathbf{G}+\boldsymbol{e}')$. Let **G'** be the generator matrix of the "demultiplied" code by the expand and sum function and $\texttt{mul}$ the multiplicity parameter, a coordinate $\dot{\boldsymbol{z}}_i$ of $\dot{\boldsymbol{z}}$ can be written as follows:

$$\forall\ 0\leqslant i<128,\ \ \dot{\boldsymbol{z}}_i=((\boldsymbol{m}\mathbf{G}')_i\oplus\boldsymbol{e}'_1)+\ldots+((\boldsymbol{m}\mathbf{G}')_i\oplus\boldsymbol{e}'_{\texttt{mul}}) \tag{5}$$

The coordinates of $\dot{\mathbf{z}}$ leave in $[\![0,\texttt{mul}]\!]$. Let's compute the probabilty distribution of $\dot{\boldsymbol{z}}_i$, namely $P(\dot{\boldsymbol{z}}_i=k)$ for any k. Either $(\boldsymbol{m}\mathbf{G}')_i=0$ with k errors, either $(\boldsymbol{m}\mathbf{G}')_i=1$ with $\texttt{mul}-k$ errors. Given that $(\boldsymbol{m}\mathbf{G}')$ is a codeword, 0 and 1 are evenly distributed. Seen as randomized Bernouilli experiments, we can compute the probability of having k errors by:

$$p_k=\binom{\texttt{mul}}{k}\cdot p_{e'}^k\cdot(1-p_{e'})^{\texttt{mul}-k} \tag{6}$$

where $p_{e'}$ refers to $P(e'_\ell=1)$ the probability that a random bit of **e'** is 1. This probability is well studied in HQC specification [19]. It follows that:

$$P(\dot{\boldsymbol{z}}_i=k)=\frac{1}{2}p_k+\frac{1}{2}p_{\texttt{mul}-k} \tag{7}$$

Distribution of intermediate values To compute the distribution of the intermediate values, we use the structure of the FHT. An intermediate vector of the FHT can be divided into blocks as follows. For a layer ℓ, we suppose the input vector of the layer is divided in blocks. Applying the butterflies on a block produces two blocks of the output vector of ℓ, one containing the addition results of the butterflies, the other one containing the subtraction results. Considering the whole input vector of the FHT as one block, each output vector of the layer ℓ is divided in 2^ℓ blocks of size $2^{7-\ell}$.

The distributions of the intermediate values are found by computing the distribution in each block in each vector, from the FHT input to the output. We called the input block, the block that determined the block we are interested in. The value distributions of the input block have already been calculated. Let $K=\{k_1,\ldots,k_N\}$ be the set of possible values of the block variables. Let F_i be the set of possible operands that gives the value k_i. The probability that a variable y of the block is equal to k_i is given by:

$$P(y=k_i)=\sum_{(a,b)\in F_i}P(x=a)\times P(x=b) \tag{8}$$

Our calculations showed that the values are normally distributed.

3.3 Using the list decoding algorithm for RS codes

The message $\boldsymbol{m}$ is the only information the adversary needs to compute the shared key. This message is the output of the decoder of the code $\mathcal{C}$. As a reminder, the decoder starts by decoding the duplicated RM code and applies the shortened RS decoder. This structure allows us to use the re-decoding strategy of Goy et al. with the Guruswami-Sudan list decoder [11] (see Section 1.1).

3.4 Simulation results

To evaluate the performance of our attack, we ran the BP algorithm on simulated leakage, as stated in Subsection 2.3. We used a multithreaded implementation of the BP algorithm written in Rust. For each value of σ, we performed 50 simulations of a random input of the FHT algorithm, with up to 500 iterations to reach convergence, stopping as soon as the distributions of the intermediates variables stopped changing.

For each noise level, we considered each simulation as successful if the algorithm converged on the right value for the message coefficient that would be decoded by the `find_peaks` function. From this criteria, we estimated the probability to recover the right value for one coefficient of the message (see dashed curves in Figure 4), and subsequently the probability to recover enough coefficients for the list decoder to be able to correct the message in order to obtain the shared secret key (see curves in Figure 4). We remind that the attack has to be performed on n_1 FHT to recover the shared key.

Figure 4 presents the success rate of the attack for the three HQC security levels. These simulations were built with noise level between 0 and 6 with a step of 0.05.

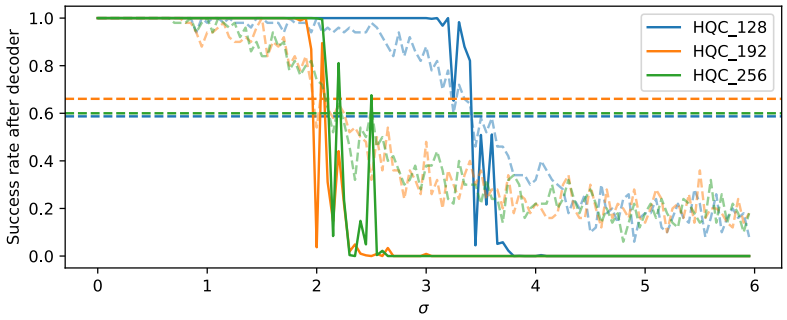

Fig. 4. Success rate of the shared key recovery attack for all HQC security levels. Non dashed curves represents the success rate of the attack. Dashed curves represents the success rate of decoding a message byte with find peaks. Dashed lines represents the proportion of messages bytes that need to be recovered for a successful decoding.

We observe that the biggest drop in noise-resistance occurs between HQC-128 and HQC-192. This phenomenon is due to an increasing of the support size of the FHT intermediate values combined with a lower error correction capability of the corresponding

list decoder. We can deduce from the simulation that our attack is able to perfectly recover the shared secret key for σ up to 3 for HQC-128, and up to 2 for the two higher security levels.

4 Recovering the secret key : two chosen ciphertext attacks

4.1 Attacker model

In this section, the purpose of the attacker is to recover the secret key $\boldsymbol{y}$. We assume that the target device performed the HQC decapsulation with a static secret key. By retrieving $\boldsymbol{y}$, the attacker is then able to decapsulate all messages encapsulated with the corresponding public key. We consider that the model described in Subsection 2.1 still stands in this case. In addition, we suppose that the attacker can send any ciphertext to the decapsulation. We are thus in a chosen ciphertext attack scenario.

Description of the chosen ciphertext attacks As identified in previous chosen ciphertexts attack [29,8], choosing the ciphertext to be $(\boldsymbol{u}, \boldsymbol{v}) = (1, \boldsymbol{v})$ leads at decoding $\boldsymbol{z} = \boldsymbol{v} - \boldsymbol{y}$. Since, this operation is performed in characteristic two, we can rewrite $\boldsymbol{z} = \boldsymbol{v} \oplus \boldsymbol{y}$. They showed how to select $\boldsymbol{v}$ in order to recover the value of the secret key $\boldsymbol{y}$. These previous attack [29,8] were able to recover the secret key with respectively around 50000 and 20000 traces. In this section, we show that we are able to build chosen ciphertexts attacks requiring either `mul` chosen ciphertexts, either a single one and a call to an ISD algorithm.

4.2 Recovering $\boldsymbol{y}$ with `mul` chosen ciphertexts

In this section, we show that the expand and sum function can be reversed by selecting `mul` ciphertexts. For a given chunk, the idea is to select `mul` ciphertexts $(\boldsymbol{v}_1, \cdots, \boldsymbol{v}_{\texttt{mul}})$ such that:

$$(\boldsymbol{v}_j)_i^{(k)} = \begin{cases} \{1\}^{128} & \text{if } j = k \\ \{0\}^{128} & \text{otherwise} \end{cases} \tag{9}$$

Therefore, with the j-th selected ciphertext, the input for the the i-th chunk of the RM decoder is $\boldsymbol{z}_j = \boldsymbol{y} \oplus \boldsymbol{v}_j$. After applying the expand and sum function, its output $\dot{\boldsymbol{z}}_j$ can be written as:

$$\dot{\boldsymbol{z}}_j = \sum_{k=1}^{\texttt{mul}} \boldsymbol{y}_i^{(k)} \oplus (\boldsymbol{v}_j)_i^{(k)} = \overline{\boldsymbol{y}_i^{(j)}} + \sum_{k=1, k\neq j}^{\texttt{mul}} \boldsymbol{y}_i^{(k)} \tag{10}$$

where $\overline{x}$ is the binary complement of x, i.e. for $x \in \mathbb{F}_2^n$, $\overline{x} \oplus x = \{1\}^n$.

We are able to implement this strategy directly with a factor graph, which outputs the secret key $\boldsymbol{y}$ without any further computation. In Figure 5, we represent the graph to perform such an attack. The graph for the multiplicity `mul` $= 5$ is the same as this one, considering 2 additional blocs.

We can prove that a number of `mul` chosen ciphertexts is enough to recover $\boldsymbol{y}$. Let us denote by $y_i^{(\ell)}$ the vector of `mul` coordinates $(\boldsymbol{y}_{i \times n_2 + l + k \times 128})_{0 \leqslant k < \texttt{mul}}$ for fixed i and

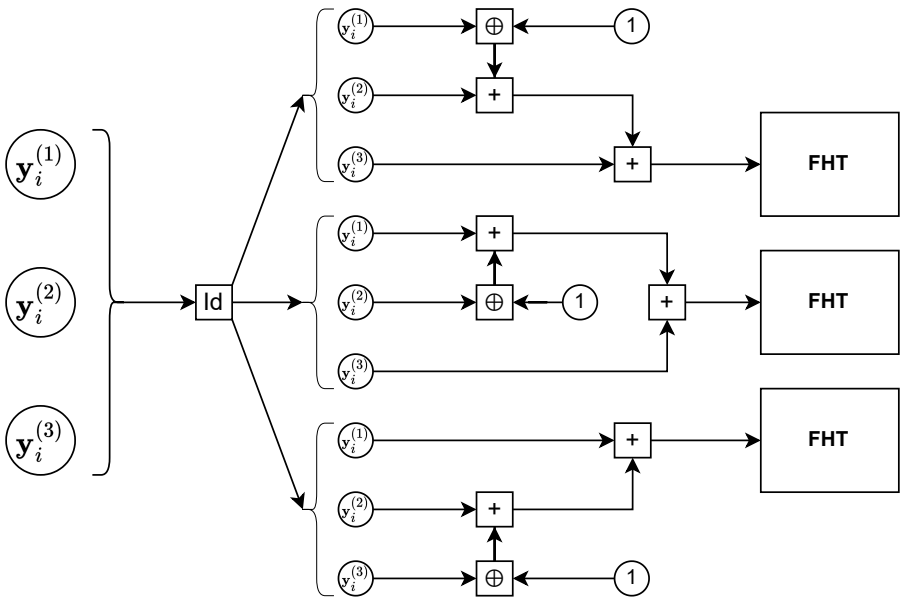

Fig. 5. Graph for the BP propagation for the chosen ciphertext attack strategy

ℓ, with $(i, \ell) \in \{0, \cdots, n_1 - 1\} \times \{0, \cdots, 127\}$. Likewise, $(v_j)_i^{(\ell)}$ refers to the vector $((\boldsymbol{v}_j)_{i \times n_2 + l + k \times 128})_{0 \leqslant k < \texttt{mul}}$. The coordinate ℓ. of the i-th FHT is given by:

$$\sum_{k=0}^{\texttt{mul}-1} (y_i)_k^{(\ell)} \oplus ((v_j)_i)_k^{(\ell)}$$

Our attack with `mul` ciphertexts $\boldsymbol{v}_j$ leads us to retrieving `mul` FHT inputs. We can show that each $y_i^{(\ell)}$ can be associated with a unique set of `mul` coordinates of the FHT inputs. We built the Table 3 of the coordinates of the `mul` FHT inputs linked to $y_i^{(\ell)}$ and $(v_j)_i^{(\ell)}$.

$(v_j)_i^{(\ell)}$ \ $y_i^{(\ell)}$	000	001	010	011	100	101	110	111
100	1	2	2	3	0	1	1	2
010	1	2	0	1	2	3	1	2
001	1	0	2	1	2	1	3	2

Table 3. Coordinates of the FHT input according to $y_i^{(\ell)}$ and $(v_j)_i^{(\ell)}$ for HQC-128

Each column of the table is unique, which proves that each $y_i^{(\ell)}$ can be recovered by an attack using 3 chosen ciphertexts. By combining all $y_i^{(\ell)}$, we retrieve $\boldsymbol{y}$. A similar table

with 5 chosen ciphertexts can be built for HQC-192 and HQC-256, showing that $\texttt{mul} = 5$ ciphertexts are sufficient.

We realized the attack with the graph presented in Figure 5. We fed the graph with simulated leakage from intermediates variables as described in Subsection 2.3. For all noise levels, we ran 100 simulations of `mul` FHT inputs produced with the probability distribution induced by our created ciphertexts. The BP algorithm was stopped when the distributions of the intermediate values no longer changed or when 100 iterations had been performed. We saw a simulation as a success only if all the intermediate values of the blocks $\boldsymbol{y}_i^{(1)}, \cdots, \boldsymbol{y}_i^{(\texttt{mul})}$ were recovered. If s_σ refers to the success rate of the 100 performed simulations for a value of σ, the success rate of the full key is given by $s_\sigma^{n_1}$. We noticed that if the simulated FHT input contains a coordinate $\geqslant 3$, the simulation often fail. The theoretical probability of these values seems to be small enough for the BP algorithm to almost never consider them to be the most probable, although they happened a non negligible number of times in our simulations. We then tried add tolerance via the priors probabilities we put in the factor graph, while keeping the FHT inputs produced with the real probability distribution associated to our chosen ciphertexts. To achieve that, we multiplied by 20 the theoretical probability that a coordinate of $\boldsymbol{y}$ is 1. The prior probabilities of the FHT intermediate values were recomputed accordingly. Its results that our attack has a success rate of 1 until a σ of 1.5 for HQC-128 and a σ of 0.5 for HQC-192 and HQC-256.

4.3 Recovering y with a single chosen ciphertext

We can go further by choosing $(\boldsymbol{u}, \boldsymbol{v}) = (1, 0)$ which leads at decoding the secret key $\boldsymbol{z} = \boldsymbol{y}$ In this case, the expand and sum function is applied to the n_2 chunks of $\boldsymbol{y}$, $\boldsymbol{y}_i = (\boldsymbol{y}_i^{(1)} | \cdots | \boldsymbol{y}_i^{(\texttt{mul})})$ and the input of the i-th FHT corresponds to $\dot{\boldsymbol{y}}_i = \sum_{k=1}^{\texttt{mul}} \boldsymbol{y}_i^{(k)}$. Decoding $\boldsymbol{y}$, which is a sparse vector with a low Hamming weight, leads to a different prior knowledge about the distribution of intermediate values inside the FHT.

Depending on the security level, we have that $p_0 = 0.988, 0.986$ or 0.987. We observe that, in this scenario, the FHT manipulates almost only zeros. We perform the attack 100 times for each noise level between 0 and 10 with a step of 0.1, with a bound to a hundred maximal iterations for the loopy BP. Results are displayed within Figure 6 for HQC-128, we obtain similar results for the next security levels.

Figure 6 shows that we are able to recover the exact value of $\dot{\boldsymbol{y}} = \left(\dot{\boldsymbol{y}}_1, \dot{\boldsymbol{y}}_2, \cdots, \dot{\boldsymbol{y}}_{n_2}\right)$ with a good accuracy up to $\sigma = 10$. But, we also observe that the success rate of the attack drops dramatically for some experiments. We have observed that this phenomenon occurs when at least one 2 is manipulated as an input to the FHT. As highlighted in Subsection 4.2, our BP has difficulties predicting values with a very small prior probability. BP then tends to consider these values as 0s, which is the most probable observation in this case. That's why in the figure, we display in the background, the proportion of each FHT input value.

A secret key that leads to manipulates only 0s or 1s as input of the FHT, that we are able to recover with our attack, will be called a weak key. We emphasize that the attack still works because the subset of weak keys is actually very dense in the total key space, and we can give the probability of randomly sampling on of them, given by the following formula:

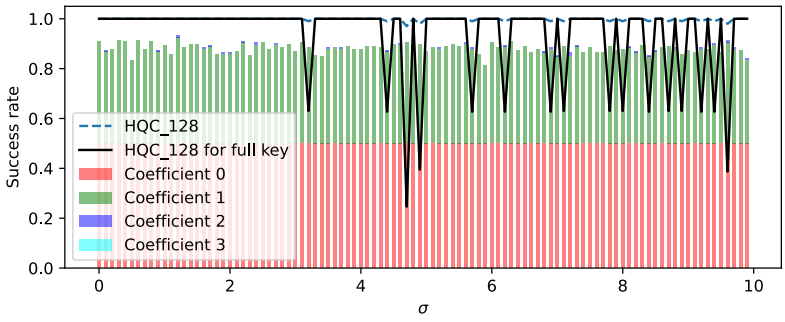

Fig. 6. Success rate of the chosen ciphertext secret key recovery attack for HQC-128.

$$\frac{\sum_{i=0}^{max(\omega,l)} \binom{l}{i} \cdot \binom{n_1 n_2/\mathtt{mul}}{\omega-i} \cdot (\mathtt{mul})^{\omega-i}}{\binom{n}{\omega}} \tag{11}$$

which gives $78,35\%$, $57,40\%$ and 55.25% of weak keys for each HQC security level.

Recovering $\boldsymbol{y}$ from the FHT input After recovering the input of the FHT, it remains to propagate this knowledge about $\dot{\boldsymbol{y}}$ to $\boldsymbol{y}$, which requires to reverse the expand and sum function.

Indeed, each 1 in $\dot{\boldsymbol{y}}$ may come from $\mathtt{mul}$ different positions in y. Furthermore, some of the support of $\boldsymbol{y}$ may be within the l truncated unused truncated bits of $\boldsymbol{y}$. Therefore, we know for sure that the support of y is contained into a set of size at most $r = \mathtt{max}\left(\omega \cdot \mathtt{mul}, (\omega-1)\cdot \mathtt{mul} + l, \cdots, (\omega - l)\cdot \mathtt{mul} + l\right)$ which is equal to $(\omega-1)\cdot \mathtt{mul} + l$ for any security level of HQC. Without any further information, we can directly exploit this information using an ISD algorithm.

We recall that HQC relies on the hardness of solving an $(\boldsymbol{s} = H\boldsymbol{e}^{\intercal}, H)$ instance of the quasi-cyclic syndrome decoding problem, where $\boldsymbol{e} = (\boldsymbol{x} \,|\boldsymbol{y})$ with $\boldsymbol{x}$ and $\boldsymbol{y}$ two vectors of length n of small Hamming weights $\mathtt{HW}(\boldsymbol{x}) = \mathtt{HW}(\boldsymbol{y}) = \omega$, and $H = (1|\boldsymbol{h})$, where $\boldsymbol{h}$ is a random vector of length n, used to generate a quasi-cyclic parity check matrix of size $n \times 2n$ (parameters of HQC can be found in Table 1).

To study the complexity of solving this instance using the knowledge on $\boldsymbol{y}$ recovered with our side-channel attack, we will use the Prange algorithm [23]. Its goal is to build an information set $\mathcal{S}$ of size n, which contains the error support. If we sample an information set randomly, then the probability that all the 2ω non null coordinates from $\boldsymbol{x}$ and $\boldsymbol{y}$ belong to this set can be approximated by:

$$p = \frac{\binom{n}{2\omega}}{\binom{2n}{2\omega}}. \tag{12}$$

The complexity of the algorithm is $\mathcal{W} \approx \frac{1}{p} n^{2.8}$, where $n^{2.8}$ is the cost of inverting a random $n \times n$ matrix over $\mathbb{F}_2$.

However, we can improve this strategy using the fact that we know for sure that the support of $\boldsymbol{y}$ belong to a set of size at most $r = (\omega - 1) \cdot \texttt{mul} + l$. We will thus select these r columns in our information set, and then sample the $n - r$ remaining ones at random in the columns corresponding to $\boldsymbol{x}$. The probability of success of an iteration of the algorithm becomes:

$$p = \frac{\binom{n-r}{\omega}}{\binom{n}{\omega}} \tag{13}$$

This value can be computed for each security level of HQC, giving $p = 0.47, 0.24$ and 0.20 for HQC 128, 192 and 256 respectively. It follows that on average, 2 ISD iterations are needed to recover the secret key in HQC-128, and 5 iterations are needed for the other security parameters.

This chosen ciphertext strategy allows to recover the value of the secret key with the analysis of a single trace. This attack can then be applied even against ephemeral keys protocols.

5 Countermeasures

5.1 Codeword Masking

Codeword masking [20] is a technique that can be used to protect the decoder of HQC againt SCA [9]. The strategy takes advantage of the linearity of the code $\mathcal{C}$. Without countermeasure, the message m is encoded to c and the decoder is applied on $c+e$. When codeword masking is performed, a message mask m' is sampled randomly and encoded to produce the mask c'. The sensitive data c is masked by feeding the decoder with $c+c'+e$. Because of the linearity of the code $\mathcal{C}$, the decoder will decode $c+c'+e$ to $m+m'$. Finally, the message is recovered by subtracting m' to the result of the decoder.

This countermeasure provides security against the attacks described in this work. Our attacks on the secret key are based on a chosen ciphertext strategy, which will be of no use since the decoder input is masked by c'. As for our attack on the shared key, it will lead us to recover $m + m'$ instead of m.

In [10], Goy et al. proposed a strategy in two steps to attack such a masked implementation, consisting of attacking both the encoder and the decoder of the RS code. The technique may be adaptable in our case, but would require a further study of the RM encoder.

5.2 Shuffling

The NTT from Kyber [6] has been the target of SASCA in [25,22,12,13,4]. Ravi et al. established two shuffling countermeasures in [27], coarse shuffling and fine shuffling, to protect the NTT from SASCA. Fine shuffling consists in randomising the order of computation of the two inputs and the two outputs of a butterfly. For each butterfly, one of the four possible combinations is randomly chosen. In the coarse shuffling countermeasure, the butterflies are shuffled within each layer independently. Both countermeasures can be applied similarly on the FHT.

Goy et al. adapted those two countermeasures against their attack on the RS decoder of HQC [10], and found them to be inefficient. However, the structure of the targeted part of the decoder makes the leaks interchangeable, which is not the case for the FHT.

In fact, the FHT is more similar to the NTT. In [13], Hermelink et al. analysed both countermeasures against SASCA on the NTT implementation of [12]. They showed that they provide protection against a classical SASCA. Nonetheless, some strategies have been proposed to weaken them. Their work serves as a warning against a false sense of security regarding shuffling countermeasures on the NTT, indicating that they may not always provide complete protection in the future. We believe that due to the resemblance between the FHT and the NTT, the FHT is likely protected by these countermeasures.

5.3 Sanity Check

A way to thwart our chosen ciphertext attacks on the secret key is to perform sanity check on the ciphertext $(\boldsymbol{u}, \boldsymbol{v})$. The principle of the countermeasure is to detect and reject malicious ciphertexts before applying the decapsulation [32,26]. Both our attacks on the secret key are stopped if the ciphertexts of the form $(\boldsymbol{u}, \boldsymbol{v}) = (1, \cdot)$ are discarded.

$\boldsymbol{u}$ and $\boldsymbol{v}$ are ciphertexts of a PKE scheme, they can be seen as random values in $\mathbb{F}_2^n$. Ciphertexts of the form $(\boldsymbol{u}, \boldsymbol{v}) = (1, \cdot)$ represent at most $\frac{1}{2^n}$ of all the ciphertexts, which is a subset of negligible size.

Conclusion

In this paper, we introduced new SASCA that can recover both the shared key and the secret key of HQC. We realized all our attacks on a simulated case with a Hamming weigh leakage model. The focus was on the FHT in the RM decoder.

The recovering of the FHT outputs by SASCA enabled us to retrieve the session key with a single trace up to a σ of 2 or 3, depending on the security level.

The advantage of our work over the first SASCA on HQC by Goy et al. [10], is that we also manage to recover the secret key. We achieved that by retrieving the FHT inputs and using a chosen ciphertext strategy. Our strongest result shows that we are able of recovering the secret key of HQC in a single attack trace, even up to very high noise levels. In practice, this attack poses a threat to the security of HQC because it targets instances with secret ephemeral keys as well, which no longer guarantees security against SCA.

We have identified several ways to protect the scheme against this new threat. The most promising countermeasure is the sanity check strategy, which prevents an attacker from using powerful chosen ciphertexts that easily give information about the secret key. Shuffling strategies adapted from the Kyber NTT, also help to reduce the strength of the attack by preventing the optimal exploitation of physical measurements. Finally, codeword masking provides security against the chosen ciphertext attacks and the shared key attack described in this work.

Further work The next step could be to study the impact of this type of attack by exploiting all the leaks that can be observed during the decapsulation of HQC. By exploiting leaks from RS and RM decoders, but also from intermediate operations, such

as expand and sum or find peaks. A complete graph of the decapsulation represents a computational challenge, most of all in multiple traces scenarios, but would allows for a very powerful error correction environment. Moreover, exploiting all possible side-channel information represents an additional strength for the attack, as the noise resistance will be increased.

Acknowledgments

This work was supported by the French National Agency in the framework of the "Investissements d'avenir" (future-oriented investments) program (ANR-10- AIRT-05) and by the defense innovation agency (AID) from the French ministry of armed forces.

References

1. Alagic, G., Apon, D., Cooper, D., Dang, Q., Dang, T., Kelsey, J., Lichtinger, J., Miller, C., Moody, D., Peralta, R., et al.: Status report on the third round of the NIST post-quantum cryptography standardization process. US Department of Commerce, NIST (2022)
2. Albrecht, M.R., Bernstein, D.J., Chou, T., Cid, C., Gilcher, J., Lange, T., Maram, V., Von Maurich, I., Misoczki, R., Niederhagen, R., et al.: Classic McEliece: conservative code-based cryptography (2022)
3. Aragon, N., Barreto, P., Bettaieb, S., Bidoux, L., Blazy, O., Deneuville, J.C., Gaborit, P., Ghosh, S., Gueron, S., Güneysu, T., et al.: BIKE: bit flipping key encapsulation (2022)
4. Assael, G., Elbaz-Vincent, P., Reymond, G.: Improving Single-Trace Attacks on the Number-Theoretic Transform for Cortex-M4. In: 2023 IEEE International Symposium on Hardware Oriented Security and Trust (HOST). pp. 111–121. IEEE (2023)
5. Bernstein, D.J., Hülsing, A., Kölbl, S., Niederhagen, R., Rijneveld, J., Schwabe, P.: The SPHINCS+ signature framework. In: Proceedings of the 2019 ACM SIGSAC conference on computer and communications security. pp. 2129–2146 (2019)
6. Bos, J., Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schanck, J.M., Schwabe, P., Seiler, G., Stehlé, D.: CRYSTALS-Kyber: a CCA-secure module-lattice-based KEM. In: 2018 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 353–367. IEEE (2018)
7. Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., Stehlé, D.: Crystals-dilithium: A lattice-based digital signature scheme. IACR Transactions on Cryptographic Hardware and Embedded Systems pp. 238–268 (2018)
8. Goy, G., Loiseau, A., Gaborit, P.: A new key recovery side-channel attack on HQC with chosen ciphertext. In: International Conference on Post-Quantum Cryptography. pp. 353–371. Springer (2022)
9. Goy, G., Loiseau, A., Gaborit, P.: Estimating the Strength of Horizontal Correlation Attacks in the Hamming Weight Leakage Model: A Side-Channel Analysis on HQC KEM. In: WCC 2022: The Twelfth International Workshop on Coding and Cryptography. p. WCC_2022_paper_48 (2022)
10. Goy, G., Maillard, J., Gaborit, P., Loiseau, A.: Single trace HQC shared key recovery with SASCA. Cryptology ePrint Archive (2023)
11. Guruswami, V., Sudan, M.: Improved decoding of Reed-Solomon and algebraic-geometric codes. In: Proceedings 39th Annual Symposium on Foundations of Computer Science (Cat. No. 98CB36280). pp. 28–37. IEEE (1998)

12. Hamburg, M., Hermelink, J., Primas, R., Samardjiska, S., Schamberger, T., Streit, S., Strieder, E., van Vredendaal, C.: Chosen ciphertext k-trace attacks on masked cca2 secure kyber. IACR Transactions on Cryptographic Hardware and Embedded Systems pp. 88–113 (2021)
13. Hermelink, J., Streit, S., Strieder, E., Thieme, K.: Adapting belief propagation to counter shuffling of NTTs. IACR Transactions on Cryptographic Hardware and Embedded Systems pp. 60–88 (2023)
14. Hofheinz, D., Hövelmanns, K., Kiltz, E.: A modular analysis of the Fujisaki-Okamoto transformation. In: Theory of Cryptography Conference. pp. 341–371. Springer (2017)
15. Kannwischer, M.J., Pessl, P., Primas, R.: Single-trace attacks on keccak. Cryptology ePrint Archive (2020)
16. Kschischang, F.R., Frey, B.J., Loeliger, H.A.: Factor graphs and the sum-product algorithm. IEEE Transactions on information theory **47**(2), 498–519 (2001)
17. MacKay, D.J.: Information theory, inference and learning algorithms. Cambridge university press (2003)
18. MacWilliams, F.J., Sloane, N.J.A.: The theory of error-correcting codes, vol. 16. Elsevier (1977)
19. Melchor, C.A., Aragon, N., Bettaieb, S., Bidoux, L., Blazy, O., Deneuville, J.C., Gaborit, P., Persichetti, E., Zémor, G.: Hamming Quasi-Cyclic (HQC) (2017)
20. Merli, D., Stumpf, F., Sigl, G.: Protecting PUF error correction by codeword masking. Cryptology ePrint Archive (2013)
21. Paiva, T.B., Terada, R.: A timing attack on the HQC encryption scheme. In: Selected Areas in Cryptography–SAC 2019: 26th International Conference, Waterloo, ON, Canada, August 12–16, 2019, Revised Selected Papers 26. pp. 551–573. Springer (2020)
22. Pessl, P., Primas, R.: More practical single-trace attacks on the number theoretic transform. In: Progress in Cryptology–LATINCRYPT 2019: 6th International Conference on Cryptology and Information Security in Latin America, Santiago de Chile, Chile, October 2–4, 2019, Proceedings 6. pp. 130–149. Springer (2019)
23. Prange, E.: The use of information sets in decoding cyclic codes. IRE Transactions on Information Theory **8**(5), 5–9 (1962)
24. Prest, T., Fouque, P.A., Hoffstein, J., Kirchner, P., Lyubashevsky, V., Pornin, T., Ricosset, T., Seiler, G., Whyte, W., Zhang, Z.: Falcon. Post-Quantum Cryptography Project of NIST (2020)
25. Primas, R., Pessl, P., Mangard, S.: Single-trace side-channel attacks on masked lattice-based encryption. In: Cryptographic Hardware and Embedded Systems–CHES 2017: 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings. pp. 513–533. Springer (2017)
26. Ravi, P., Chattopadhyay, A., D'Anvers, J.P., Baksi, A.: Side-channel and fault-injection attacks over lattice-based post-qu'ntum schemes (Kyber, Dilithium): Survey and new results. ACM Transactions on Embedded Computing Systems (2022)
27. Ravi, P., Poussier, R., Bhasin, S., Chattopadhyay, A.: On Configurable SCA Countermeasures Against Single Trace Attacks for the NTT: A Performance Evaluation Study over Kyber and Dilithium on the ARM Cortex-M4. In: Security, Privacy, and Applied Cryptography Engineering: 10th International Conference, SPACE 2020, Kolkata, India, December 17–21, 2020, Proceedings 10. pp. 123–146. Springer (2020)
28. Schamberger, T., Holzbaur, L., Renner, J., Wachter-Zeh, A., Sigl, G.: A power side-channel attack on the reed-muller reed-solomon version of the HQC cryptosystem. In: International Conference on Post-Quantum Cryptography. pp. 327–352. Springer (2022)
29. Schamberger, T., Renner, J., Sigl, G., Wachter-Zeh, A.: A power side-channel attack on the CCA2-secure HQC KEM. In: Smart Card Research and Advanced Applications: 19th

International Conference, CARDIS 2020, Virtual Event, November 18–19, 2020, Revised Selected Papers 19. pp. 119–134. Springer (2021)
30. Veyrat-Charvillon, N., Gérard, B., Standaert, F.X.: Soft analytical side-channel attacks. In: Advances in Cryptology–ASIACRYPT 2014: 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, ROC, December 7-11, 2014. Proceedings, Part I 20. pp. 282–296. Springer (2014)
31. Wafo-Tapa, G., Bettaieb, S., Bidoux, L., Gaborit, P., Marcatel, E.: A practicable timing attack against HQC and its countermeasure. Cryptology ePrint Archive (2019)
32. Xu, Z., Pemberton, O., Roy, S.S., Oswald, D., Yao, W., Zheng, Z.: Magnifying side-channel leakage of lattice-based cryptosystems with chosen ciphertexts: The case study of kyber. IEEE Transactions on Computers **71**(9), 2163–2176 (2021)
33. Zhou, Y., Yu, Y., Standaert, F.X., Quisquater, J.J.: On the need of physical security for small embedded devices: a case study with comp128-1 implementations in sim cards. In: Financial Cryptography and Data Security: 17th International Conference, FC 2013, Okinawa, Japan, April 1-5, 2013, Revised Selected Papers 17. pp. 230–238. Springer (2013)