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], Antoine Loiseau[2]

[1*]XLIM - University of Limoges, Limoges, France.
[2]Univ. Grenoble Alpes, CEA, Leti, F-38000 Grenoble, France.
[3]Inria, 91120 Palaiseau, France.
[4]IP Paris, 91120 Palaiseau, France.

*Corresponding author(s). E-mail(s): chloe.baisse@unilim.fr;

Abstract

Soft Analytical Side Channel Attacks (SASCA) are a powerful family of Side Channel Attacks (SCA) that allows the recovery of 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. Our attacks are realized on simulations. 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, enabling attacks targeting both the secret key and shared key. 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 different chosen ciphertext attacks. One of them requires a single trace and is successful until high noise levels.

Keywords: Soft Analytical Side Channel Attacks (SASCA), Belief Propagation (BP), Hamming Quasi-Cyclic (HQC), Post-Quantum Cryptography (PQC), Single Trace Attacks, Chosen Ciphertext Attacks (CCA)

1 Introduction

Hamming Quasi-Cyclic (HQC) [1] is a code-based Key Encapsulation Mechanism (KEM), and a candidate for the National Institute of Standards and Technology (NIST)'s contest for the standardization of post-quantum cryptosystems. After three rounds, NIST selected the KEM CRYSTALS-Kyber [2] as well as the signature schemes CRYSTALS-Dilithium [3], FALCON [4] 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 [6], BIKE [7] and HQC. During the course of the contest, the security of the candidates has been tested against Side-Channel Attacks (SCA), and post-quantum schemes were found to be vulnerable to this kind of attacks. Protection against SCA is one of the metric used to evaluate candidates in the NIST's process [8]. 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.

Concerning HQC, a first version of the cryptosystem that uses BCH codes has been targeted by two Timing Attacks (TA) in 2019 [9] and 2020 [10], as well as a chosen ciphertext attack [11] in 2020. During the third round of the NIST contest, a new version of the cryptosystem based on concatenated Reed-Muller (RM) and Reed-Solomon (RS) codes was released.

A message recovery attack with a single electromagnetic trace by Goy et al. [12], 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 [13, 14] where the support of the secret key $\mathbf{y}$ is recovered thanks to an analysis of the decoder's behaviour according to its inputs.

Soft Analytical Side Channel Attacks (SASCA) were first proposed in 2014 by Veyrat-Charvillon et al. on the AES cryptosystem [15]. 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 time and memory complexities. SASCA allowed authors to drastically reduce the number of side-channel measurements compared to a template attack, even down to a single-trace for low noise levels. Since this work on AES, several cryptographic standards have been attacked by SASCA. In 2020, Kannwischer et al. applied SASCA for different kind of scenarios [17], on the Keccak hash function that often deals with ephemeral secrets. SASCA was also used to attack the authenticated lightweight encryption algorithm ASCON [18] in 2022 [19] and 2023 [20].

Related Work

The lattice-based cryptosystem Kyber [2], standardized by NIST, has also been a target of SASCA with five attacks [21–25]. All of them focused on the Number Theoretic Transform (NTT) or its inverse, used in the scheme to perform polynomial multiplications efficiently. The NTT procedure is 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. This structure makes heavy use of load and store operations, which are known to produce a high amount of side-channel leakage.

The first attack was proposed in 2017 by Primas et al. [21] and targeted both the unmasked and the masked implementations. 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. Compared to the attack of 2017, the noise tolerance is increased for the unmasked implementation but is smaller for the masked one. In 2021, Hamburg et al. [23] applied a k-trace SASCA ($k \in \{2, 3, 4\}$) using a chosen ciphertext strategy in order to have a sparse input linked to the secret. They are able to reach higher noises for the masked implementations than the two previous attacks. Countermeasures against single trace attacks on the NTT have been proposed in 2020 by Ravi et al. [26], including fine shuffling and coarse shuffling. In 2023, these two shuffling countermeasures have been tested by Hermelink et al. [24] against the Hamburg et al. attack [23]. While they were found to be efficient, some tools have been introduced to weaken them. Assael et al. [25] attacked an optimized implementation of Kyber for ARM Cortex M4 in 2023. They improved the runtime of the BP algorithm by using message pruning.

Additionally, Kyber has been the target of several side channel attacks that use chosen ciphertexts specifically built to increase the leakages of the targeted operation. In 2021, Xu et al. [27] used this strategy on the NTT or on the message encoding, which allows them to recover Kyber's secret key using from 4 to 960 traces depending on the attacked implementation. Soon after, Ravi et al. [28] exploited ciphertext malleability properties to perform message recovery attacks on lattice-based NIST post-quantum candidates, including Kyber. They also improved the attack of Xu et al. by making it more generic regarding lattice-based public key schemes, and by reducing the number of attack traces from 8 to 6 for the first security level of Kyber. An attack using leakages of the Barret Reduction of Kyber's decapsulation step by Sim et al. [29] successfully retrieves the secret key with 6 to 12 chosen ciphertexts according to the security level or the implementation targeted.

In 2023, Goy et al. [30] led the first SASCA on HQC, 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 high noise level. They show that a masking countermeasure does not provide security against their attack, although the noise resistance is lower. They also adapted the shuffling countermeasures [26] used to protect Kyber 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 simulated 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 derive 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 simulated leaks of the computation of the Fast Hadamard Transform (FHT) allow to recover the HQC shared key. This attack can benefit from the re-decoding technique, introduced by Goy et al. [12, 30] 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.

2 Background

2.1 Hamming Quasi-Cyclic

2.1.1 General scheme

Hamming Quasi-Cyclic (HQC) [1] is a code-based Key Encapsulation Mechanism (KEM) that allows to share a session key to enable later communications 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 $[n, k]$ and 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) [31] 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 $\mathcal{R}_\omega = \{\mathbf{x} \in \mathcal{R} \ ; \ \mathrm{HW}(\mathbf{x}) = \omega\}$.

Algorithm 1 KeyGen

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

Algorithm 2 Encrypt

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

Algorithm 3 Decrypt

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

In HQC-KEM, the encryption function is de-randomised thanks to a seed derived from the message (the hash of the latter). 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
$\mathbf{m} \xleftarrow{\$} \mathbb{F}_2^k$
$\theta \leftarrow \mathcal{G}(\mathbf{m})$
$\mathbf{c} \leftarrow \text{Encrypt}(p_k, \mathbf{m}, \theta)$
$K \leftarrow \mathcal{K}(\mathbf{m}, \mathbf{c})$ ▷ shared key
$\mathbf{d} \leftarrow \mathcal{H}(\mathbf{m})$
return $(\mathbf{c}, \mathbf{d})$

Algorithm 5 Decapsulate

Input: $s_k, \mathbf{c}, \mathbf{d}$
$\mathbf{m} \leftarrow \text{Decrypt}(s_k, \mathbf{c})$
$\theta' \leftarrow \mathcal{G}(\mathbf{m}')$
$\mathbf{c}' \leftarrow \text{Encrypt}(p_k, \mathbf{m}', \theta')$
if $\mathbf{c} \neq \mathbf{c}'$ or $\mathbf{d} \neq \mathcal{H}(\mathbf{m}')$ **then**
  return $\perp$
else
  return $\mathcal{K}(\mathbf{m}, \mathbf{c})$
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 [1].

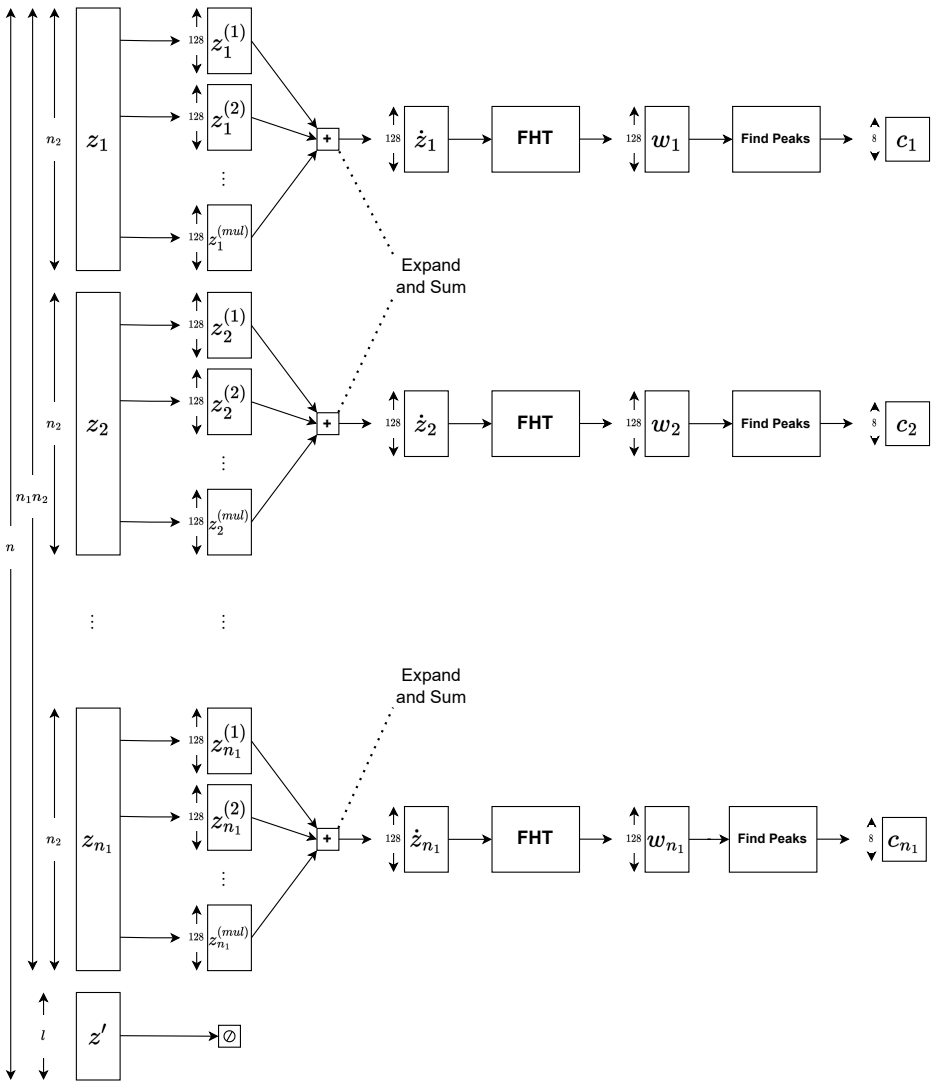

Fig. 1 Structure of the HQC expanded Reed-Muller decoder.

2.1.2 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. This gives the parameters $[n_2, k_2]$ of Table 1. 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 [32].

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 [1] of April 2023, the FHT is implemented following Algorithm 6.

Algorithm 6 Fast Hadamard Transform from [1].

```
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;
    }
}
```

2.1.3 Decoding $v - uy$

In HQC, the decoding is applied on the vector $\mathbf{z} = \mathbf{v} - \mathbf{uy}$ (see Algorithm 3). This 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 $\mathbf{z} = \mathbf{v} - \mathbf{uy}$ 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 (see 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 $n_2 = 128 \times \texttt{mul}$, namely $\mathbf{z}_i = \left(\mathbf{z}_i^{(1)} \mid \mathbf{z}_i^{(2)} \mid \cdots \mid \mathbf{z}_i^{(\texttt{mul})}\right)$ and returns as output $\dot{\mathbf{z}}_i = \sum_{k=1}^{\texttt{mul}} \mathbf{z}_i^{(k)}$, the sum of each block (this operation is the natural sum over $\mathbb{N}$). The expand and sum algorithm then returns a vector $\dot{\mathbf{z}}_i$ of length 128 whose each value leaves in $[0, \texttt{mul}]$. This vector is the input of the FHT, which outputs a vector $\mathbf{w}_i \in [-64 \cdot \texttt{mul}, 128 \cdot \texttt{mul}]^{128}$. 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 $\mathbf{c}_i$, which is the final output of the RM decoder.

2.1.4 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. [12, 30]. This strategy aims at correcting the errors made during the attacks by 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 has 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 [33] 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.

2.2 Soft Analytical Side Channel Attacks

Belief Propagation (BP) [16] 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 [34], Markov random fields [35], and the decoding of Low Density Parity Check codes [36].

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) [15] go further by connecting the gathered side-channel information in a graphical model, linking the intermediate values according to the performed mathematical operations. By applying BP on this model, the attacker is able to efficiently compute the marginal distributions of the intermediate variables, which will lead to finding the marginal distribution of the secret.

2.2.1 The factor graph

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

A variable node represents an intermediate variable of the algorithm. It is meant to contain the marginal probability distribution of the variable computed by BP.

Factor nodes are of two types. Some of them embody an operation of the algorithm. Their goal is to check the operation's consistency with any combination of intermediate variable values involved. They return 1 if the values are coherent with the operation and 0 otherwise. The nodes of the intermediate variables taking part in the operation are linked by an edge to the factor node. There are also observational factor nodes that hold a probability distribution associated to a variable. Their purpose is to contain the probability distribution of the variable found by a template attack. It is also possible to add observational factor nodes that carry the prior probability distribution of the variable. An edge exists between the variable node and its associated observational factor nodes.

2.2.2 The Belief Propagation algorithm

The Belief Propagation (BP) algorithm [16] aims at computing the marginals of every intermediate variable of the attacked function. In other words, BP calculates the probability of each possible value for every intermediate variable given the information present in the factor graph. It is done thanks to a message-passing principle where 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 [37]. 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. The goal is to compute the marginal probability of every variable x_n, given by

$$Z_n(x_n) = \sum_{\{x_n\}_{n' \neq n}} \prod_{m=1}^{M} f_m(\mathbf{x}_m) \tag{1}$$

To achieve that, two types of messages are passed along the edges of the factor graph.

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

$$q_{n \to m}(x_n) = \prod_{m' \in \mathcal{M}(n) \setminus \{m\}} r_{m' \to n}(x_n) \tag{2}$$

(ii) A message sent by a factor f_m to a variable x_n is given by:

$$r_{m \to n}(x_n) = \sum_{\mathbf{x}_m \setminus \{x_n\}} \left(f_m(\mathbf{x}_m) \prod_{n' \in \mathcal{N}(m) \setminus \{n\}} q_{n' \to m}(x_{n'}) \right) \tag{3}$$

When all messages have been computed and transmitted, the marginal of a variable x_n is found by normalizing the product of the incoming messages at the associated node:

$$Z_n(x_n) = \prod_{m \in \mathcal{N}(m)} r_{m \to n}(x_n) \tag{4}$$

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

2.2.3 Loopy-BP

Cryptographic operations are often modeled by cyclic factor graphs. In that case, we use an iterative algorithm called loopy-BP. Ideally, the message-passing procedure is repeated until the marginals between two iteration stop changing. Nevertheless, in cyclic factor graphs, the marginals are not necessary correct and the convergence may never happen. A maximum number of iterations in case the convergence is not reached must be defined. In addition, as the marginals are often approximate, a threshold on the maximal statistical change of the variables' distributions can be specified to detect convergence. The results of the loopy-BP often happened to be sufficient for most applications, including the computation of marginal distributions for side-channel attacks.

3 Attacking HQC with SASCA

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

3.1 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. We assume 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 butterflies can be identified thanks to simple supervised leakage assessment.

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. The analysis of the simulations allows to evaluate the resistance of our attack to noise and to show the correction capacity provided by BP.

Numerous preceding studies indicate that translating the results of a simulated attack into a practical attack on a simple target, such as the STM32F407, while not straightforward, is feasible [38, 39]. In the case of SASCA, this transposition is realized in several papers [17, 20, 22, 30].

3.2 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:

$$\forall\ 0 \leqslant i \leqslant 63,\ \begin{cases} y_i &= x_{2i} + x_{2i+1} \\ y_{i+64} &= x_{2i} - x_{2i+1} \end{cases} \tag{5}$$

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

$$f_+(a,b,c) = \begin{cases} 1 \text{ if } a+b=c \\ 0 \quad \text{otherwise} \end{cases} \qquad f_-(a,b,c) = \begin{cases} 1 \text{ if } a-b=d \\ 0 \quad \text{otherwise} \end{cases} \tag{6}$$

This butterfly forms numerous and short cycles in the complete FHT factor graph. We choose to apply the clustering strategy [22, 23] 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:

$$f_{bf}(a,b,c,d) = \begin{cases} 1 \text{ if } a+b=c \text{ and } a-b=d \\ 0 \qquad\qquad\quad \text{otherwise} \end{cases} \tag{7}$$

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 lead to increased performance [22].

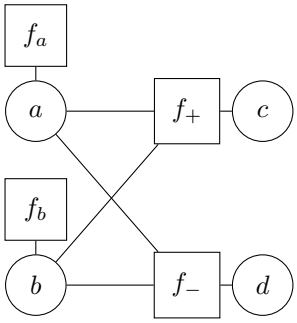

Fig. 2 Direct butterfly FG.

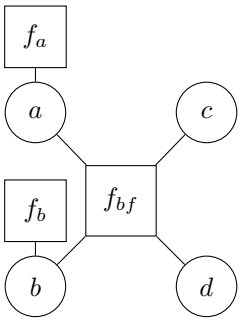

Fig. 3 Chosen butterfly FG.

3.3 Leakage simulations

To realize our simulations, we rely on the Hamming weight leakage model [40]. This model is suitable for describing leakages when the targeted circuit uses a load/store architecture, i.e. when data is sent via buses between memory and registers. The measurement $\mathrm{tr}(s)$ of an intermediate variable s is defined by:

$$\mathrm{tr}(s) = \alpha\,\mathrm{HW}(s) + \mathcal{N}(\beta, \sigma^2). \tag{8}$$

Increasing the value of σ means increasing the noise level.

Theorically, template matching is used to derive $P(\mathrm{HW}(s)|tr(s))$. Eventually, the probability distribution of s is computed as follows:

$$P(s|tr(s)) := P(s|\,\mathrm{HW}(s)) \cdot P(\mathrm{HW}(s)|tr(s)) \tag{9}$$

In practice, we select $\alpha = 1$ and $\beta = 0$ without loss of generality and simulate the value r of a linear classifier prediction when the value s is manipulated with the following equation:

$$P(r \mid s) = P(X = r) \tag{10}$$

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

3.4 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 $\mathbf{y}$. The FHT of HQC is thus a highly interesting target for SASCA. In the following two sections, we show how an attacker can retrieve these two secrets thanks to SASCA.

4 Recovering the shared key

4.1 Attacker model

We suppose that the attacker follows the model described in Subsection 3.1. Since our attack aims at recovering the shared key, which is an ephemeral secret, we only consider a single trace attack scenario. Then, 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. Moreover, the shared key information is contained in 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 find peaks function on this output, we retrieve a byte c_i of the erroneous RS codeword (see Figure 1). We repeat the attack n_1 times to recover every c_i. We are now able to use the re-decoding strategy of Goy et al. [12] with the Guruswami-Sudan list decoder [33] (see subsubsection 2.1.4). We obtain the message $\mathbf{m}$, that is the only information needed to compute the shared key.

4.2 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 nodes. The BP algorithm will combine these nodes with the observational nodes containing the probability distributions found from the leakages, which will increase the accuracy of the results.

We explain how to compute the probability distributions of the FHT input vector. The duplicated RM decoder takes $\mathbf{v}-\mathbf{uy}$ as input. It is a codeword of $\mathcal{C}$ with an error **e'**. Indeed, $\mathbf{v}-\mathbf{uy}$ is equal to $\mathbf{mG}+\mathbf{xr}_2+\mathbf{yr}_1+\mathbf{e}$ that we rewrite $\mathbf{mG}+\mathbf{e}'$ with $\mathbf{e}'=\mathbf{xr}_2+\mathbf{yr}_1+\mathbf{e}$. The input of the FHT can be denoted as $\dot{\mathbf{z}} = \mathit{expand_and_sum}(\mathbf{mG}+\mathbf{e}')$. Let **G'** be the generator matrix of the "demultiplied" code by the expand and sum function and `mul` the multiplicity parameter, a coordinate $\dot{\mathbf{z}}_i$ of $\dot{\mathbf{z}}$ can be written as follows:

$$\forall\ 0 \leqslant i < 128,\ \dot{\mathbf{z}}_i = ((\mathbf{mG}')_i \oplus \mathbf{e}'_1) + \ldots + ((\mathbf{mG}')_i \oplus \mathbf{e}'_{\mathtt{mul}}) \tag{11}$$

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

$$p_k = \binom{\mathtt{mul}}{k} \cdot p_{e'}^k \cdot (1-p_{e'})^{\mathtt{mul}-k} \tag{12}$$

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 [1]. It follows that:

$$P(\dot{\mathbf{z}}_i = k) = \frac{1}{2}p_k + \frac{1}{2}p_{\mathtt{mul}-k} \tag{13}$$

4.3 Simulation results

To evaluate the performance of our attack, we run the BP algorithm on simulated leakage, as stated in Subsection 3.3. We use a custom multithreaded implementation of the BP algorithm written in Rust, but the BP implementation by SCALib [41] is also suitable for the experiments. For each value of σ, we perform 50 simulations of a random input of the FHT algorithm, with a maximum of 500 iterations of the loopy-BP algorithm, stopping as soon as the distributions of the intermediates variables stop changing.

For each noise level, we consider each simulation as successful if the algorithm converges on the right value for the message coefficient that would be decoded by the find peaks function. From this criteria, we estimate 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 on the n_1 FHT for the three HQC security levels. These simulations are built with values of σ between 0 and 6 with a step of 0.05.

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

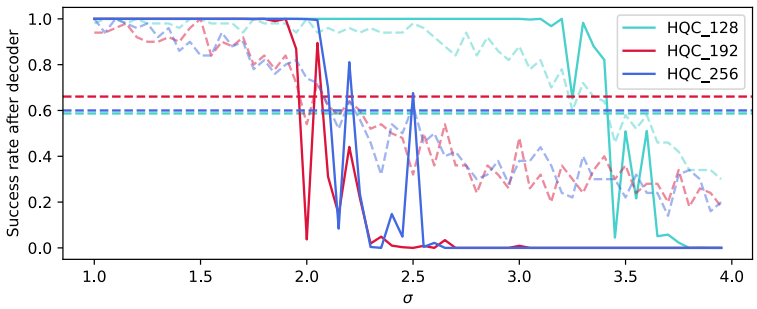

Fig. 4 Success rate of the shared key recovery attack for all HQC security levels. Non dashed curves represents the success rate of the attack. Dashed curves represent the success rate of decoding a message byte with find peaks. Dashed lines represent the proportion of messages bytes that need to be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

corresponding list decoder. Besides, we note that significant variations of the attack success rate occur when the recovery rate of a byte after find peaks is close to the proportion of correct message bytes required for a successful decoding. Whenever the success rate of a byte decoding falls below this proportion, it results in a consequent drop in the success rate of the attack.

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.

5 Recovering the secret key : two chosen ciphertext attacks

5.1 Attacker model

In this section, the purpose of the attacker is to recover the secret key $\mathbf{y}$. We assume that the target device performed the HQC decapsulation with a static secret key. By retrieving $\mathbf{y}$, the attacker is then able to decapsulate all messages encapsulated with the corresponding public key. We consider that the model described in Subsection 3.1 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.

5.2 Description of the chosen ciphertext attacks

As identified in previous chosen ciphertext attacks [11, 14], choosing the ciphertext to be $(\mathbf{u}, \mathbf{v}) = (1, \mathbf{v})$ leads at decoding $\mathbf{z} = \mathbf{v} - \mathbf{y}$. Since, this operation is performed in characteristic two, we can rewrite $\mathbf{z} = \mathbf{v} \oplus \mathbf{y}$. They showed how to select $\mathbf{v}$ in order to recover the value of the secret key $\mathbf{y}$. These previous attacks [11, 14] 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 ciphertext attacks requiring either `mul` chosen ciphertexts, either a single one and a call to an ISD algorithm.

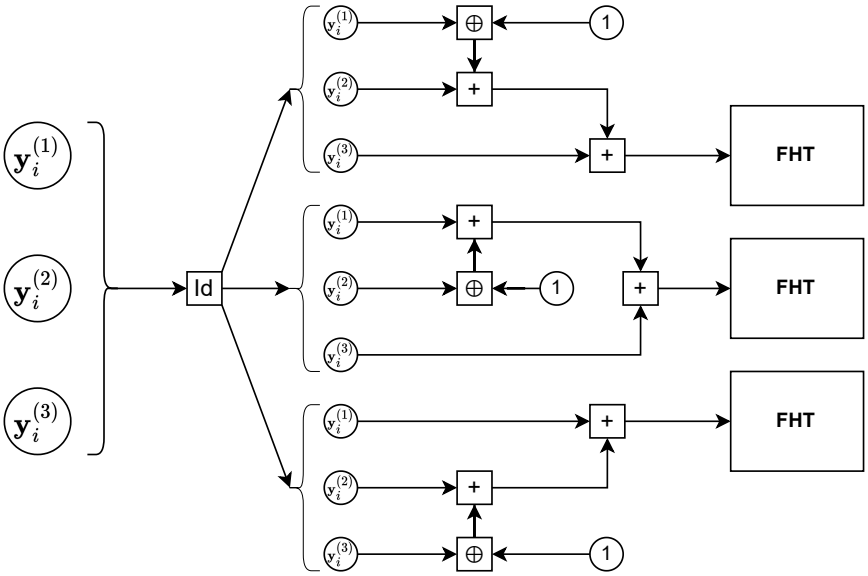

Fig. 5 Graph for the BP propagation for the chosen ciphertext attack strategy.

5.3 Recovering 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 $(\mathbf{v}_1, \cdots, \mathbf{v}_{\texttt{mul}})$ such that:

$$(\mathbf{v}_j)_i^{(k)} = \begin{cases} \{1\}^{128} & \text{if } j = k \\ \{0\}^{128} & \text{otherwise} \end{cases} \tag{14}$$

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

$$\dot{\mathbf{z}}_j = \sum_{k=1}^{\texttt{mul}} \mathbf{y}_i^{(k)} \oplus (\mathbf{v}_j)_i^{(k)} = \overline{\mathbf{y}_i^{(j)}} + \sum_{k=1, k\neq j}^{\texttt{mul}} \mathbf{y}_i^{(k)} \tag{15}$$

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 SASCA, and recover the secret key $\mathbf{y}$. In Figure 5, we represent the graph to perform such an attack. The graph for the multiplicity $\texttt{mul} = 5$ is the same as this one, considering 2 additional blocks.

We can prove that a number of `mul` chosen ciphertexts is enough to recover $\mathbf{y}$. Let us denote by $\hat{\mathbf{y}}_i^{(\ell)}$ the vector of `mul` coordinates $(\mathbf{y}_{i\times n_2+l+k\times 128})_{0\leqslant k<\texttt{mul}}$ for fixed i and ℓ, with $(i, \ell) \in \{0, \cdots, n_1 - 1\} \times \{0, \cdots, 127\}$. Likewise, $(\hat{\mathbf{v}}_j)_i^{(\ell)}$ refers to the vector $((\mathbf{v}_j)_{i\times n_2+l+k\times 128})_{0\leqslant k<\texttt{mul}}$. The coordinate ℓ. of the i-th FHT is given by:

$$\sum_{k=0}^{\texttt{mul}-1} (\hat{\mathbf{y}}_i)_k^{(\ell)} \oplus ((\hat{\mathbf{v}}_j)_i)_k^{(\ell)} \tag{16}$$

Our attack with `mul` ciphertexts $\mathbf{v}_j$ leads us to retrieving `mul` FHT inputs. We can show that each $\hat{\mathbf{y}}_i^{(\ell)}$ 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 $\hat{\mathbf{y}}_i^{(\ell)}$ and $(\hat{\mathbf{y}}_j)_i^{(\ell)}$.

$(\hat{\mathbf{v}}_j)_i^{(\ell)}$ \ $\hat{\mathbf{y}}_i^{(\ell)}$	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 $\hat{\mathbf{y}}_i^{(\ell)}$ can be recovered by an attack using 3 chosen ciphertexts. By combining all $\hat{\mathbf{y}}_i^{(\ell)}$, we retrieve $\mathbf{y}$. A similar table with 5 chosen ciphertexts can be built for HQC-192 and HQC-256, showing that `mul` $= 5$ ciphertexts are sufficient.

We realized the attack with the graph presented in Figure 5. We fed the graph with simulated leakages from intermediates variables as described in Subsection 3.3. 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. 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 value $\geqslant 3$, 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 adding 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 $\mathbf{y}$ is 1. The prior probability distributions of the FHT intermediate values were recomputed accordingly. It 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.

5.4 Recovering y with a single chosen ciphertext

We can go further by choosing $(\mathbf{u}, \mathbf{v}) = (1, 0)$ which leads at decoding the secret key $\mathbf{z} = \mathbf{y}$. In this case, the expand and sum function is applied to the n_2 chunks of $\mathbf{y}$, $\mathbf{y}_i = (\mathbf{y}_i^{(1)} | \cdots | \mathbf{y}_i^{(\texttt{mul})})$ and the input of the i-th FHT corresponds to $\dot{\mathbf{y}}_i = \sum_{k=1}^{\texttt{mul}} \mathbf{y}_i^{(k)}$. Decoding $\mathbf{y}$, 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 value of σ 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.

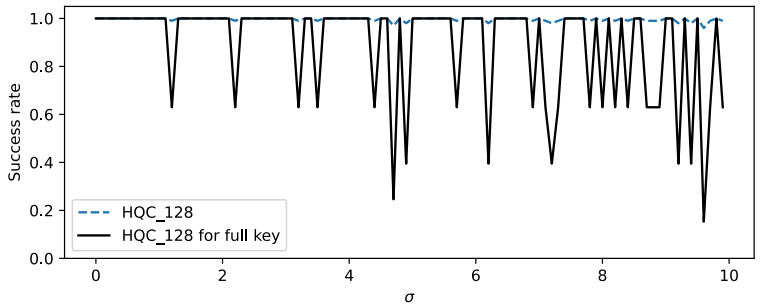

Fig. 6 Success rate of the chosen ciphertext secret key recovery attack for HQC-128.

Figure 6 shows that we are able to recover the exact value of $\dot{\mathbf{y}} = (\dot{\mathbf{y}}_1, \dot{\mathbf{y}}_2, \cdots, \dot{\mathbf{y}}_{n_2})$ 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 in particular when at least one 2 is manipulated as an input to the FHT. As highlighted in Subsection 5.3, 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 is why in the Figure 7, we display the success rate of the attack when the input of the FHT contains only 0s and 1s. While some drops of the success rate, their number is reduced and they are less significant. Furthermore, they become nonexistent for values of $\sigma \leqslant 3$.

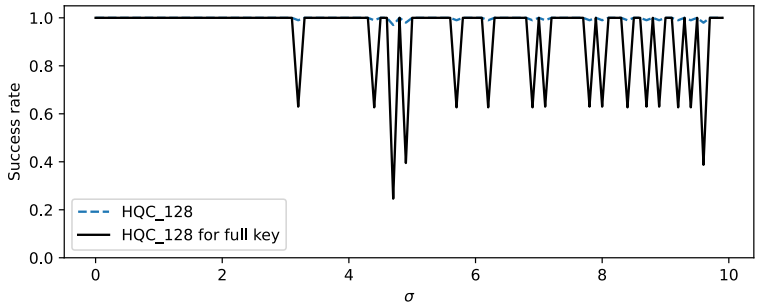

Fig. 7 Success rate of the chosen ciphertext secret key recovery attack for HQC-128 where the input of the FHT contains only 0s or 1s.

A secret key that leads to the manipulation of 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:

$$\frac{\sum_{i=0}^{max(\omega,l)} \binom{l}{i} \cdot \binom{n_1 n_2/\mathtt{mul}}{\omega-i} \cdot (\mathtt{mul})^{\omega-i}}{\binom{n}{\omega}} \tag{17}$$

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

5.4.1 Recovering y from the FHT input

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

Indeed, each 1 in $\dot{\mathbf{y}}$ may come from $\mathtt{mul}$ different positions in y. Furthermore, some of the support of $\mathbf{y}$ may be within the l truncated unused truncated bits of $\mathbf{y}$. Therefore, we know for sure that the support of $\mathbf{y}$ 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 $(\mathbf{s} = \mathbf{H}\mathbf{e}^\intercal, \mathbf{H})$ instance of the quasi-cyclic syndrome decoding problem, where $\mathbf{e} = (\mathbf{x}|\mathbf{y})$ with $\mathbf{x}$ and $\mathbf{y}$ two vectors of length n of small Hamming weights $\mathrm{HW}(\mathbf{x}) = \mathrm{HW}(\mathbf{y}) = \omega$, and $H = (1|\mathbf{h})$, where $\mathbf{h}$ 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 $\mathbf{y}$ recovered with our side-channel attack, we will use the Prange algorithm [42]. 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 $\mathbf{x}$ and $\mathbf{y}$ belong to this set can be approximated by:

$$p = \frac{\binom{n}{2\omega}}{\binom{2n}{2\omega}}. \tag{18}$$

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 $\mathbf{y}$ belong to a set of size at most $r = (\omega-1) \cdot \mathtt{mul} + l$. 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 $\mathbf{x}$. The probability of success of an iteration of the algorithm becomes:

$$p = \frac{\binom{n-r}{\omega}}{\binom{n}{\omega}} \tag{19}$$

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.

6 Countermeasures

6.1 Codeword Masking

Codeword masking [43] is a technique that can be used to protect the decoder of HQC againt SCA [12]. 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.

Goy et al. [30] 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.

6.2 Shuffling

The NTT from Kyber [2] has been the target of several SASCA [21–25]. Ravi et al. [26] established two shuffling countermeasures, 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 [30], 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. Hermelink et al. [24] analysed both countermeasures against SASCA on the NTT implementation of [23]. 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.

6.3 Sanity Check

A way to thwart our chosen ciphertext attacks on the secret key is to perform sanity check on the ciphertext $(\mathbf{u}, \mathbf{v})$. The principle of the countermeasure is to detect and reject malicious ciphertexts before applying the decapsulation [27, 44]. Our attacks on the secret key used ciphertexts $(\mathbf{u}, \mathbf{v})$ such as $\mathbf{u} = 1$, but they work with any $\mathbf{u}$ of Hamming weight 1 as $\mathbf{u}$ simply rotates $\mathbf{y}$ in that case. Then, our chosen ciphertext attacks are stopped if the ciphertexts $(\mathbf{u}, \mathbf{v})$ such as $\mathbf{u}$ has Hamming weight 1 are discarded. They represent $\frac{n}{2^n}$ of all the ciphertexts, which is a subset of negligible size. More generally, possible variants of our attacks can be countered by rejecting pairs $(\mathbf{u}, \mathbf{v})$ where $\mathbf{u}$ and $\mathbf{v}$ both have low Hamming weights.

7 Conclusion

In this paper, we introduced three new SASCA on HQC: one targeting the shared key, and two that recover the secret key with a chosen ciphertext strategy. We realized all our attacks on a simulated case with a Hamming weight leakage model, focusing on the FHT in the RM decoder. 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. A possible 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 side-channel leakages in other parts of the decoder could bring insightful information in order to increase the noise resilience of the attack.

Declarations

Funding

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), by the defense innovation agency (AID) from the French ministry of armed forces, and by the PIA4 X7PQC project.

References

[1] 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)

[2] 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 (2018). https://doi.org/10.1109/EuroSP.2018.00032 . IEEE

[3] 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, 238–268 (2018) https://doi.org/10.46586/tches.v2018.i1.238-268

[4] 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)

[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). https://doi.org/10.1145/3319535.3363229

[6] 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)

[7] 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)

[8] 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)

[9] Wafo-Tapa, G., Bettaieb, S., Bidoux, L., Gaborit, P., Marcatel, E.: A Practicable Timing Attack Against HQC and its Countermeasure. Cryptology ePrint Archive (2019) https://doi.org/10.3934/amc.2020126

[10] 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 (2020). https://doi.org/10.1007/978-3-030-38471-5_22 . Springer

[11] 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 (2021). https://doi.org/10.1007/978-3-030-68487-7_8 . Springer

[12] 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, pp. 2022–48 (2022)

[13] 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 (2022). https://doi.org/10.1007/978-3-031-17234-2_16 . Springer

[14] 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 (2022). https://doi.org/10.1007/978-3-031-17234-2_17 . Springer

[15] Veyrat-Charvillon, N., Gérard, B., Standaert, F.-X.: Soft Analytical Aide-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 (2014). https://doi.org/10.1007/978-3-662-45611-8_15 . Springer

[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) https://doi.org/10.1109/18.910572

[17] Kannwischer, M.J., Pessl, P., Primas, R.: Single-Trace Attacks on Keccak. IACR Transactions on Cryptographic Hardware and Embedded Systems **2020**(3), 243–268 (2020) https://doi.org/10.46586/tches.v2020.i3.243-268

[18] Dobraunig, C., Eichlseder, M., Mendel, F., Schläffer, M.: Ascon v1. 2: Lightweight Authenticated Encryption and Hashing. Journal of Cryptology **34**, 1–42 (2021) https://doi.org/10.1007/s00145-021-09398-9

[19] Luo, S., Wu, W., Li, Y., Zhang, R., Liu, Z.: An Efficient Soft Analytical Side-Channel Attack on Ascon. In: International Conference on Wireless Algorithms, Systems, and Applications, pp. 389–400 (2022). https://doi.org/10.1007/978-3-031-19208-1_32 . Springer

[20] You, S.-C., Kuhn, M.G., Sarkar, S., Hao, F.: Low Trace-Count Template Attacks on 32-bit Implementations of ASCON AEAD. IACR Transactions on Cryptographic Hardware and Embedded Systems **2023**(4), 344–366 (2023) https:

//doi.org/10.46586/tches.v2023.i4.344-366

[21] 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 (2017). https://doi.org/10.1007/978-3-319-66787-4_25 . Springer

[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 (2019). https://doi.org/10.1007/978-3-030-30530-7_7 . Springer

[23] Hamburg, M., Hermelink, J., Primas, R., Samardjiska, S., Schamberger, T., Streit, S., Strieder, E., Vredendaal, C.: Chosen Ciphertext k-TraceAttacks on Masked CCA2 Secure Kyber. IACR Transactions on Cryptographic Hardware and Embedded Systems, 88–113 (2021) https://doi.org/10.46586/tches.v2021.i4.88-113

[24] Hermelink, J., Streit, S., Strieder, E., Thieme, K.: Adapting Belief Propagation to Counter Shuffling of NTTs. IACR Transactions on Cryptographic Hardware and Embedded Systems, 60–88 (2023) https://doi.org/10.46586/tches.v2023.i1.60-88

[25] 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 (2023). https://doi.org/10.1109/HOST55118.2023.10133270 . IEEE

[26] 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 (2020). https://doi.org/10.1007/978-3-030-66626-2_7 . Springer

[27] Xu, Z., Pemberton, O., Roy, S.S., Oswald, D., Yao, W., Zheng, Z.: Magnifying Side-Channel Leakage of Lattice-Lased Cryptosystems with Chosen Ciphertexts: The Case Study of Kyber. IEEE Transactions on Computers **71**(9), 2163–2176 (2021) https://doi.org/10.1109/TC.2021.3122997

[28] Ravi, P., Bhasin, S., Roy, S.S., Chattopadhyay, A.: On Exploiting Message Leakage in (few) NIST PQC Candidates for Practical Message Recovery Attacks. IEEE Transactions on Information Forensics and Security **17**, 684–699 (2021)

[29] Sim, B.-Y., Park, A., Han, D.-G.: Chosen- Ciphertext Clustering Attack on CRYSTALS-KYBER Using the Side-Channel leakage of Barrett Reduction. IEEE

Internet of Things Journal **9**(21), 21382–21397 (2022)

[30] Goy, G., Maillard, J., Gaborit, P., Loiseau, A.: Single Trace HQC Shared Key Recovery with SASCA. IACR Transactions on Cryptographic Hardware and Embedded Systems **2024**(2), 64–87 (2024) https://doi.org/10.46586/tches.v2024.i2.64-87

[31] Hofheinz, D., Hövelmanns, K., Kiltz, E.: A Modular Analysis of the Fujisaki-Okamoto Transformation. In: Theory of Cryptography Conference, pp. 341–371 (2017). https://doi.org/10.1007/978-3-319-70500-2_12 . Springer

[32] MacWilliams, F.J., Sloane, N.J.A.: The Theory of Error-Correcting Codes vol. 16. Elsevier, Amsterdam (1977)

[33] 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 (1998). https://doi.org/10.1109/SFCS.1998.743426 . IEEE

[34] Pearl, J.: Reverend Bayes on Inference Engines: A Distributed Hierarchical Approach. In: Probabilistic and Causal Inference: The Works of Judea Pearl, pp. 129–138 (2022). https://doi.org/10.1145/3501714.3501727

[35] Isham, V.: An Introduction to Spatial Point Processes and Markov Random Fields. International Statistical Review/Revue Internationale de Statistique, 21–43 (1981) https://doi.org/10.2307/1403035

[36] Gallager, R.: Low-Density Parity-Check Codes. IRE Transactions on information theory **8**(1), 21–28 (1962) https://doi.org/10.7551/mitpress/4347.001.0001

[37] MacKay, D.J.: Information Theory, Inference and LearningAlgorithms. Cambridge university press, Cambridge (2003)

[38] Brier, E., Clavier, C., Olivier, F.: Correlation Power Analysis with a Leakage Model. In: Cryptographic Hardware and Embedded Systems-CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings 6, pp. 16–29 (2004). Springer

[39] 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 (2013). https://doi.org/10.1007/978-3-642-39884-1_20 . Springer

[40] Mangard, S., Oswald, E., Popp, T.: Power Analysis Attacks: Revealing the Secrets of Smart Cards vol. 31. Springer, Berlin (2008)

[41] Cassiers, G., Bronchain, O.: Scalib: A side-channel analysis library. Journal of Open Source Software **8**(86), 5196 (2023) https://doi.org/10.21105/joss.05196

[42] Prange, E.: The Use of Information Sets in Decoding Cyclic Codes. IRE Transactions on Information Theory **8**(5), 5–9 (1962) https://doi.org/10.1109/TIT.1962.1057777

[43] Merli, D., Stumpf, F., Sigl, G.: Protecting PUF Error Correction by Codeword Masking. Cryptology ePrint Archive (2013)

[44] Ravi, P., Chattopadhyay, A., D'Anvers, J.P., Baksi, A.: Side-Shannel and —Fault-Injection Attacks Over Lattice-Based Post-Quantum Schemes (Kyber, Dilithium): Survey and New Results. ACM Transactions on Embedded Computing Systems (2022) https://doi.org/10.1145/3603170