A New Key Recovery Side-Channel Attack on HQC with Chosen Ciphertext

Guillaume Goy, Antoine Loiseau, Phlippe Gaborit

A New Key Recovery Side-Channel Attack on HQC with Chosen Ciphertext

Guillaume Goy[1,2], Antoine Loiseau[1], and Philippe Gaborit[2]

[1] Univ. Grenoble Alpes, CEA, Leti, MINATEC Campus, F-38054 Grenoble, France
{guillaume.goy,antoine.loiseau}@cea.fr
[2] XLIM, University of Limoges
gaborit@unilim.fr

Abstract. Hamming Quasi-Cyclic (HQC) is a code-based candidate of NIST post-quantum standardization procedure. The decoding steps of code-based cryptosystems are known to be vulnerable to side-channel attacks and HQC is no exception to this rule. In this paper, we present a new key recovery side-channel attack on HQC with chosen ciphertext. Our attack takes advantage of the reuse of a static secret key on a micro-controller with a physical access. The goal is to retrieve the static secret key by targeting the Reed-Muller decoding step of the decapsulation and more precisely the Hadamard transform. This function is known for its diffusion property, a property that we exploit through side-channel analysis. The side-channel information is used to build an Oracle that distinguishes between several decoding patterns of the Reed-Muller codes. We show how to query the Oracle such that the responses give a full information about the static secret key. Experiments show that less than 20.000 electromagnetic attack traces are sufficient to retrieve the whole static secret key used for the decapsulation. Finally, we present a masking-based countermeasure to thwart our attack.

Keywords: HQC · Reed-Muller Codes · Chosen Ciphertext Attack · Side-Channel Attack · Post-Quantum Cryptography

1 Introduction

The interest for Post-Quantum Cryptography (PQC) increased with the quantum computers threat to classic cryptography schemes like RSA [20]. The research is promoted by the National Institute of Standards and Technology (NIST) who launched a call for proposal [16] in 2016 with the aim to standardize new signature and Key Encapsulation Mechanism (KEM) schemes. NIST moves closer to making standardization decisions and aims to precisely measuring the security of the schemes, including Side-Channel Attacks (SCA) and their countermeasures. Thus, the security against SCA and the cost and performance of side-channel protection could be criteria for standards selection [3].

Hamming Quasi-Cyclic (HQC) [1,2,4] is a promising candidate of the fourth round of the PQC NIST contest. Unlike the McEliece construction [15] and

derivates, the security of HQC is not related to hiding the structure of an error correcting code. In HQC, the structure of the decoding codes is publicly known and the security can be reduced to the Quasi-Cyclic version of the well know Syndrome Decoding (SD) problem [2,5,24].

Nowadays cryptographic schemes are assessed to be theoretically secure, for HQC, the security comes from two sides. On the one hand, the IND-CCA2 security is provided by the transformation of a IND-CPA scheme with a Fujisaki-Okamoto like transform [7,8,11]. This point guarantees the security against malicious adversaries who would make a diverted use of the scheme. On the other hand, finding the secret key is impossible given the reduction of security to a known NP-hard problem [5]. However, implementation of a secure scheme in constrained devices, such as micro-controller, can still be vulnerable to physical attacks.

Side-Channel Attacks (SCA) [12,13], introduced by P. Kocher in 1996, are non-invasive physical attacks with aim to exploit side-channel leakage (timing, power consumption, electromagnetic radiation, execution time, heat, sound ...). Since their introduction, SCA have a long history of success in extracting secret information (such as secret key or message) of cryptographic algorithms [6,19,26]. The leakage is statistically dependent on the intermediate variables that are processed and this side-channel information can be exploited to extract secret information.

Related Works SCA already targeted the HQC scheme in various ways. In 2019 and 2020, the first version of HQC based on BCH codes was attacked by Timing attacks (TA). These TA [17,25] use a correlation between the weight of the decoded error and the computation time of the decoder. As a result, HQC authors' team proposed a constant time implementation for decoding BCH codes to mitigate these TA.

In 2021, a novel TA [10,23] targeted the RMRS version of HQC. This TA uses the rejection sampling construction to attack both HQC and BIKE. Indeed, the sampling of a vector of small Hamming weight ω is performed by randomly choosing its support. ω locations are sampled and sometimes collisions occur at these locations which, leads to rejecting the vector and sampling another one from the beginning. However, all the randomness is generated with a seed (see Algorithm 4) which is derived from the exchanged message used to compute the shared key. This observation leads to a relation between the run-time of the rejection sampling and the exchanged message. This relation is strong enough to extract information about the secret key in HQC with a chosen ciphertext strategy.

In 2022, an horizontal SCA [9] used the Decryption Failure Rate (DFR) of HQC by targeting Reed-Solomon (RS) decoding. Indeed, the low DFR implies that the Reed-Muller decoder almost always decodes all the errors. This leads to an error-free codeword decoding by the RS decoder. By studying the behaviour of the RS decoder and using a better decoder for the RS codes in order to correct

side-channel induced errors, authors are able to recover the exchanged message in a single trace.

In 2020 and 2022, Schamberger et al. [22,21] proposed two chosen ciphertext attacks (CCA) based on a side-channel Oracle able to determine whenever an error is corrected by the HQC decoder through a supervised approach. These attacks are possible despite the IND-CCA2 security because the side-channel distinguisher is performed before the re-encryption phase, during the decoding part. The first attack [22] targets the BCH version of HQC, they chose the ciphertext in order to create a single error for the BCH decoder, which can be seen by the Oracle. They are able to recover a large part of the possible keys in HQC with a secret key support (non-zero locations) research. By a complex chosen ciphertext attack, they are able to deduce information about the support of the secret key which is used during the decapsulation. Authors adapted their power side-channel attack to the Reed-Muller Reed-Solomon version of HQC [21], leading to a complex but functional attack on the new version of HQC. These attacks are a serious threat to the HQC security and no countermeasure have been proposed to thwart these attacks neither in the first paper, nor in the second. In this context, finding a new distinguisher on the HQC decoding procedure allows to build the same kind of attacks.

Our Contributions In this paper, we propose a simpler key recovery side-channel attack with a chosen ciphertext strategy targeting the new RMRS version of HQC. We are able to retrieve a static secret key of HQC by building a chosen ciphertext attack with a less complex queries selection process. The main idea is to build queries in order to create collisions with the secret key support, changing the decoding behaviour. We show how to construct a new simple Oracle on the RM decoding step that is able to determine the number of corrected errors. This Oracle is based on a supervised side-channel approach with the used of a Linear Discriminant Analysis. The knowledge of the secret key is not required for the Oracle's training stage, allowing a training phase on a clone device and reducing the number of attack measurements. We build an Oracle of 120.000 training traces and use 50 attack traces per bit to create an attack with 100% accuracy. A divide and conquer strategy allows at recovering the whole decoding static secret key with less than 20.000 attack traces. Our attack can be avoided by a masking-based countermeasure applied on the Oracle target function.

Paper Organization The paper is organized as follows: In Section 2, we recall the HQC framework and the main algorithms useful for the understanding of the attack. Section 3 is devoted to the description of our chosen ciphertext attack, showing how to build the queries in order to recover the secret key using a decoding Oracle. In Section 4, we describe the construction and evaluation of the Oracle based on side-channel measurements. Then we present our results and give the attack complexity. We present a simple masking based countermeasure in Section 5 to thwart our attack before concluding.

2 Hamming Quasi-Cyclic (HQC)

2.1 HQC overview

HQC [2] is a code-based post-quantum resistant Key Encapsulation Mechanism (KEM). Unlike other code-based cryptosystems, the security of HQC does not rely on hiding the structure of an error correcting code. The security is guaranteed by a random double circulant code with a reduction to the well-studied Quasi-Cyclic Syndrome Decoding Problem (QCSD) [2,5,24]. HQC uses another code $\mathcal{C}$ with an efficient decoder $\mathcal{C}.\mathsf{Decode}$ that is publicly known. Neither the security of the scheme nor the decryption capability depend on the knowledge of $\mathcal{C}.\mathsf{Decode}$. A classic construction is to turn a Public Key Encryption (PKE) scheme into a KEM. HQC-PKE is fully described by three algorithms (see Algorithms 1, 2 and 3 in Figure 1). Considering $\mathcal{R} = \mathbb{F}_2[X]/(X^n - 1)$ the ambient space with n a primitive prime given as parameter and $\mathcal{R}_\omega$ the space restriction to words of Hamming weight ω.

Algorithm 1 Keygen

Input: param
Output: $(\mathsf{pk}, \mathsf{sk})$

1: $\mathbf{h} \xleftarrow{\$} \mathcal{R}$
2: $(\mathbf{x}, \mathbf{y}) \xleftarrow{\$} \mathcal{R}_\omega^2$
3: $\mathbf{s} = \mathbf{x} + \mathbf{hy}$
4: $\mathsf{pk} = (\mathbf{h}, \mathbf{s})$
5: $\mathsf{sk} = (\mathbf{x}, \mathbf{y})$

Algorithm 2 Encrypt

Input: $(\mathsf{pk}, \mathbf{m})$, param
Output: ciphertext $\mathbf{c}$

1: $\mathbf{e} \xleftarrow{\$} \mathcal{R},\ \mathrm{wt}(\mathbf{e}) = \omega_e$
2: $(\mathbf{r}_1, \mathbf{r}_2) \xleftarrow{\$} \mathcal{R}_{\omega_r}^2$
3: $\mathbf{u} = \mathbf{r}_1 + \mathbf{hr}_2$
4: $\mathbf{v} = \mathbf{m}G + \mathbf{sr}_2 + \mathbf{e}$
5: $\mathbf{c} = (\mathbf{u}, \mathbf{v})$

Algorithm 3 Decrypt

Input: $(\mathsf{sk}, \mathbf{c})$
Output: $\mathbf{m}$

1: $\mathbf{m} = \mathcal{C}.\mathsf{Decode}(\mathbf{v} - \mathbf{uy})$

Fig. 1: HQC-PKE Algorithms [1,2]

A quantum adapted Fujisaki-Okamoto transformation [7,8] called the Hofheinz-Hövelmanns-Kiltz (HHK) transformation [11] turns the PKE into a KEM and allows HQC-KEM scheme to reach IND-CCA2 security. The main idea of such a construction is the re-encryption during the decapsulation that prevent from chosen ciphertext attack (CCA). The KEM IND-CCA2 property is guaranteed given that the PKE has been proved IND-CPA (see HQC specifications [2] for details). HQC KEM algorithms [1] are described with Algorithms 4 and 5 in Figure 2.

As mentioned earlier, the formal security of HQC does not rely on the chosen publicly known code. Therefore, this code can be chosen at the convenience of the developer. Authors of HQC propose the use of a concatenated code with a duplicated Reed-Muller (RM) code for internal code and a Reed-Solomon (RS) code for the external one. Formally, the internal code is a $[n_1, k_1, d_1]$ code over $\mathbb{F}_q$ and an external code a $[n_2, k_2, d_2]$ code, with $q = 2^{k_2}$. To encode with

Algorithm 4 Encaps

Input: $\mathsf{pk} = (\mathbf{h}, \mathbf{s})$
Output: $(\mathbf{c}, \mathbf{d})$

1: $\mathbf{m} \xleftarrow{\$} \mathbb{F}_2^k$
2: $\theta = \mathcal{G}(\mathbf{m})$ ▷ seed
3: $\mathbf{c} = \mathsf{Encrypt}(\mathbf{m}, \mathsf{pk}, \theta)$
4: $K = \mathcal{K}(\mathbf{m}, \mathbf{c})$
5: $\mathbf{d} = \mathcal{H}(\mathbf{m})$
6: **return** $(\mathbf{c}, \mathbf{d})$

Algorithm 5 Decaps

Input: $\mathbf{c}, \mathbf{d}, \mathsf{sk}, \mathsf{pk}$
Output: shared key K or $\perp$

1: $\mathbf{m'} = \mathsf{Decrypt}(\mathbf{c}, \mathsf{sk})$
2: $\theta' = \mathcal{G}(\mathbf{m}')$ ▷ seed
3: $\mathbf{c}' = \mathsf{Encrypt}(\mathbf{m}', \mathsf{pk}, \Theta')$ ▷ re-encrypt
4: **if** $\mathbf{c} \neq \mathrm{c}$ or $\mathbf{d} \neq \mathcal{H}(\mathbf{m}')$ **then**
5: **return** $\perp$
6: **else**
7: **return** $K = \mathcal{K}(\mathbf{m}, \mathbf{c})$
8: **end if**

Fig. 2: HQC-KEM Algorithms [1,2]. (Same key gen. as PKE version, see Fig. 1)

a concatenated construction, we first encode a message of length k_1 with the external code to obtain an intermediate codeword of length n_1 over $\mathbb{F}_q = \mathbb{F}_{2^{k_2}}$. The internal code can be independently applied on each of the n_1 elements of $\mathbb{F}_q$, leading to encode n_1 times with the internal code, obtaining a final codeword of length $n_1 n_2$ (see Figure 3).

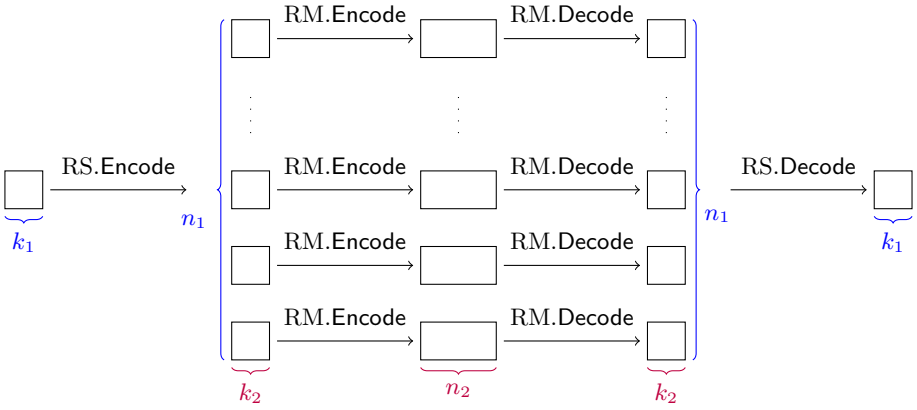


Fig. 3: Simplified HQC Concatenated RMRS Codes Framework

The decoder is then a double decoder that decodes the internal code first and then the external. The decoding procedure is the same as the encoder and the first operation is to decode a $n_1 n_2$ sized codeword with n_1 independent decoding steps applied on blocs of size n_2. Our attack only targets these RM decoding

steps and, for the sake of clarity, we will only describe the RM construction in this paper.

2.2 Decoding Reed-Muller Codes

HQC uses the same RM code RM$(1, 7)$, which is a $[128, 8, 64]$ code over $\mathbb{F}_2$, regardless of the chosen security level. Furthermore, each bit is duplicated 3 or 5 times (see Figure 2), adding multiplicity to the codewords, to obtain codes of parameters $[384, 8, 192]$ or $[640, 8, 320]$. These RM codes of order 1 are seen as Hadamard codes and can be decoded using a Fast Hadamard Transform (FHT). The decoding procedure is always the same, composed of three main algorithms in the June 2020 HQC reference implementation [1]. The procedure is composed by the following main steps:

1. Removing the multiplicity of codewords with the expand and sum function (see Algorithm 6).
2. Apply the Fast Hadamard Transform (FHT) (see Algorithm 7).
3. Recover the message with the find peaks function (see Algorithm 8).

The first step goal is to remove the multiplicity of the codeword by adding in $\mathbb{N}$ bit to bit each repetition (see Algorithm 6). The result is an expanded codeword of length 128 leaving in $[\![0, 3]\!]$ or $[\![0, 5]\!]$ depending on the value of the multiplicity mul.

Algorithm 6 Expand and sum

Input: codeword $\mathbf{c}$ and the multiplicity mul.
Output: expanded codeword $\mathbf{c}'$

$\mathbf{c}' = \mathbf{0} \in \mathbb{N}^{128}$
for $i \in [\![0, \mathsf{mul}]\!]$ **do**
  for $j \in [\![0, 128]\!]$ **do**
    $\mathbf{c}'[j]$ += $\mathbf{c}[128 \times i + j]$
  end for
end for
return c'

Fast Hadamard Transform (FHT) The FHT is a generalized discrete Fourier Transform applied to expand codeword (see Algorithm 7). In practice, this function is equivalent to multiplying the expand codeword with an Hadamard matrix. Indeed, MacWilliams and Sloane [14] showed that the weight distribution of the cosets can be determined by the application of an Hadamard transform. This allows to decode with a maximum likelihood strategy, finding the distance between a received message and every codewords. For a RM$(1, m)$ code, the Hadamard matrix to choose is H_{2^m} which can be described recursively (see Equation (1)).

$$H_{2m} = \begin{pmatrix} H_m & H_m \\ H_m & -H_m \end{pmatrix}, \; H_1 = 1 \tag{1}$$

Applying such a vector matrix multiplication would require $2^m \times 2^m$ additions and subtractions. Fortunately, H_{2^m} can be written as a product of m $2^m \times 2^m$ sparse matrices with only two non-zeros elements per column (see Equation 2). This observation allows to change the vector matrix multiplication by m vector and sparse matrix multiplication.

$$H_{2^m} = M_{2^m}^{(1)} M_{2^m}^{(2)} \cdots M_{2^m}^{(m)},\ M_{2^m}^{(i)} = I_{2^{m-i}} \otimes H_2 \otimes I_{2^{i-1}},\ 1 \leq i \leq m \qquad (2)$$

with I_n the identity matrix of size $n \times n$ and $\otimes$ the Kronecker product. That is the reason why the Algorithm 7 is composed of a for loop with argument $m = 7$.

Algorithm 7 Fast Hadamard Transform (FHT)

Input: expanded codeword $\mathbf{c}$ and the multiplicity mul.
Output: expanded codeword transformed structure $\mathbf{c}$

1: **for** $pass \in [\![0, 6]\!]$ **do**
2: **for** $i \in [\![0, 63]\!]$ **do**
3: $\mathbf{d}[i] = \mathbf{c}[2i] + \mathbf{c}[2i + 1]$
4: $\mathbf{d}[i + 64] = \mathbf{c}[2i] - \mathbf{c}[2i + 1]$
5: **end for**
6: $\mathsf{swap}(\mathbf{d}, \mathbf{c})$ ▷ copy $\mathbf{d}$ in $\mathbf{c}$ and $\mathbf{c}$ in $\mathbf{d}$
7: **end for**
8: $\mathbf{c}'[0]$ −= $64 * \mathsf{mul}$
9: **return** $\mathbf{c}$

In our RM$(1, 7)$ case, this transformation returns a vector of length 128. The last function Find Peaks permits to finish the decoding. Among this vector, the argument of the absolute maximum gives the 7 least significant bits of the decoded message. The most significant bit is given by the sign of this maximum (see Algorithm 8).

Algorithm 8 Find Peaks

Input: expanded codeword transformed structure $\mathbf{c}$
Output: message $\mathbf{m}$

1: $(\text{peak\_value}, \text{peak\_abs\_value}, \text{peak\_pos}) = (0, 0, 0)$
2: **for** $i \in [\![0, 123]\!]$ **do**
3: $(t, t_{\text{abs}}) = (\mathbf{c}[i], \mathsf{absolute}(\mathbf{c}[i]))$
4: **if** $t_{\text{abs}} >$ peak_abs_value **then**
5: $(\text{peak\_value}, \text{peak\_abs\_value}, \text{peak\_pos}) = (t, t_{\text{abs}}, i)$
6: **end if**
7: **end for**
8: peak_pos += $128 * (\text{peak\_value} > 0)$ ▷ Setting the msb
9: **return** peak_pos

3 Theoretical combined Chosen Ciphertext and Side-Channel Attacks

In this section, we present a new attack to recover the secret key $\mathbf{y} \in \mathbb{F}_2^n$. The main operation during the decapsulation part of HQC is decoding the erroneous codeword $\mathbf{v} - \mathbf{uy}$. Then the knowledge of the $\mathbf{y}$ part of the secret key is enough to decapsulate.

Attack Scenario We consider a physical access to a device performing the HQC decapsulation with a static secret key. We assume that we can submit any ciphertext to the device. Our goal is to retrieve this key and then be able to decapsulate any message encapsulated by the associated public key. Our attack exploits the side-channel leakage to create an Oracle that is able to distinguish between several decoding patterns. We use the chosen ciphertext attack construction to send appropriate ciphertexts to the static secret key decapsulation chip. The electromagnetic measurements during the decapsulation constitute queries we can give to the Oracle.

We show how to build queries that allows to fully recover $\mathbf{y}$. First of all, notice that choosing the special ciphertext $(\mathbf{u}, \mathbf{v}) = (\mathbf{1}, \mathbf{0})$ leads to decode the secret key $\mathbf{y}$. Vector $\mathbf{y}$ is a sparse vector, with a small Hamming weight $\mathrm{wt}(\mathbf{y}) = \omega$. As a result, the RM decoder manipulates almost only zeros which is decoding into $\mathbf{0}$. This ciphertext is rejected by the re-encryption phase and the decapsulation returns a random vector as shared key for the IND-CCA2 security. This security could prevent our attack, however this is not an issue, given that our distinguisher does not depend on the output of the decapsulation but on the SCA leakage. Indeed, before the re-encryption, this ciphertext is manipulated by HQC decryption algorithms, among which the RM decoder, which give us all the necessary information to recover the secret key.

The RM decoder is independently applied on n_1 codewords blocs of size n_2 (see Figure 3), individually retrieving each of the n_1 bloc of $\mathbf{y}$ leads at recovering $\mathbf{y}$. Our Oracle allows to recognize the number of errors corrected in each bloc. The idea is to vary the number of corrected errors by choosing the value of $\mathbf{v}$. The goal is to find $\mathbf{v} = \mathbf{y}$ which leads to decode $\mathbf{v} - \mathbf{y} = \mathbf{0}$. We show a strategy to effectively achieve this result.

The Oracle behaviour depends on the Hamming weight of the bloc to be decoded. Then, we study the support distribution of $\mathbf{y}$ among its different blocs.

3.1 Support Distribution of y

The support $\mathsf{Supp}(\mathbf{x})$ of a vector $\mathbf{x} = (x_0, \cdots, x_n)$ is the location of its non-zero coordinates (see Equation (3)). If $\mathbf{x}$ is seen as a binary vector, its support is exactly the locations of the ones and the knowledge of the support is equivalent to the knowledge of the vector.

$$\mathsf{Supp}(\mathbf{x}) = \{i \in \mathbb{Z} \mid x_i \neq 0\} \tag{3}$$

The vector $\mathbf{y}$ has a length of n bits which is the smallest primitive prime number greater than $n_1 n_2$. The primitive prime n is used for the ambient space in order to thwart structural attacks. However, only the $n_1 n_2$ first bits, corresponding to the length of the concatenated code, are used during the decoding code, making the last $l = n - n_1 n_2$ truncated bits useless. Then, $\mathbf{y} \in \mathbb{F}_2^n$ can be seen as the concatenation of two vectors $\mathbf{y} = (\mathbf{y}', \mathbf{y}'')$ with $\mathbf{y}' \in \mathbb{F}_2^{n_1 n_2}$ and $\mathbf{y}'' \in \mathbb{F}_2^l$. This particularity prevents us for recovering any information about $\mathsf{Supp}(\mathbf{y}'')$. Fortunately, these bits are not relevant for the decoding step, and setting them to $\mathbf{0}$ is sufficient for a successful decoding. Furthermore, in almost all cases, $\mathrm{wt}(\mathbf{y}'') = 0$ (see Figure 1) and in other cases, $\mathrm{wt}(\mathbf{y}'')$ is close too zero with a high probability. The probability $\mathbb{P}(\mathrm{wt}(\mathbf{y}'') = k)$ can be approximated by:

$$Q_k := \mathbb{P}(\mathrm{wt}(\mathbf{y}'') = k) \cong \binom{\omega}{k} p^k (1-p)^{\omega - k} \tag{4}$$

where $\omega = \mathrm{wt}(\mathbf{y})$ and p the probability to draw with replacement a bit in $\mathbf{y}'$ which is equal to $p = \dfrac{|\mathbf{y}'|}{|\mathbf{y}|}$.

λ	n	$n_1 n_2$	ω	Q_0	Q_1	Q_2	$Q_{\geq 2}$
128	17.669	17.664	66	$98,15\%$	$1,83\%$	$0,02\%$	$\leq 10^{-3}\%$
192	35.851	35.840	100	$96,98\%$	$2,98\%$	$0,05\%$	$\leq 10^{-3}\%$
256	57.637	57.600	131	$91,93\%$	$7,73\%$	$0,32\%$	$\leq 10^{-2}\%$

Table 1: A few parameters of HQC and Support Probability Distribution[3] between $\mathbf{y}'$ and $\mathbf{y}''$ following Equation (4)

Recovering $\mathbf{y}'$ is enough to call the attack successful. Nevertheless, strategies exist to deduce the last l bits, for a complete recovery of $\mathbf{y}$. This low Hamming weight distribution allows to build an exhaustive research for the last $l = n - n_1 n_2$ bits of $\mathbf{y}$. Alternatively, Schamberger et al. [22], proposes a method in Section 3.3 of their paper to recover $\mathsf{Supp}(\mathbf{y}'')$ given the knowledge of $\mathsf{Supp}(\mathbf{y}')$ in polynomial time.

Support Distribution of $\mathbf{y}'$ The vector $\mathbf{y}'$ lives in $\mathbb{F}_2^{n_1 n_2}$ and the external decoder manipulates the codeword by bloc of size n_1, we can rewrite:

$$\mathbf{y}' = \left(\mathbf{y}'_0, \mathbf{y}'_1, \cdots, \mathbf{y}'_{n_1 - 1}\right) \text{ and for all } i,\ \mathbf{y}'_i \in \mathbb{F}_2^{n_2} \tag{5}$$

For our attack, the worst case is when $\mathbf{y}'$ is full weight, i.e. $\mathrm{wt}(\mathbf{y}') = \mathrm{wt}(\mathbf{y})$ which happens when $\mathrm{wt}(\mathbf{y}'') = 0$ with a high probability (see Figure 1). Indeed, this case increases the probability of having blocs with high Hamming weight. Later we will see that our distinguisher can only distinguish blocs with Hamming

[3] For each line, the sum is not equal to one because of the chosen approximation

weight up to τ. Since the support distribution of $\mathbf{y}$ is almost always in this unfavorable case, we will only consider it for the following.

The Reed-Muller decoder manipulates each bloc $\mathbf{y}'_i$ independently, we calculate the probability P_k such as a randomly sampled bloc $\mathbf{y}'_i$ has an Hamming weight of k (see Equation (6) and Figure 2).

$$P_k := \mathbb{P}\left(\mathrm{wt}(\mathbf{y}'_i) = k \;\middle|\; \mathbf{y} \xleftarrow{\$} \mathcal{R}_\omega,\ i \xleftarrow{\$} [\![0, n_1 - 1]\!]\right) \tag{6}$$

λ	mul.	n_1	n_2	ω	P_0	P_1	P_2	P_3	P_4	$P_{\geq 5}$
128	3	46	384	66	$23,44\%$	$34,38\%$	$24,83\%$	$11,77\%$	$4,12\%$	$1,45\%$
192	5	56	640	100	$16,50\%$	$30,00\%$	$27,00\%$	$16,04\%$	$7,07\%$	3.40%
256	5	90	640	131	$23,14\%$	$34,06\%$	$24,87\%$	$12,02\%$	$4,32\%$	$1,59\%$

Table 2: A few parameters of HQC and Support Probability Distribution[3] among the blocs of $\mathbf{y}'$ following Equation (6).

From Figure 2, we observe that the small Hamming weight blocs are mostly represented. Furthermore, it is relatively rare to sample blocs of weight greater than or equal to 5. For the following, as an approximation, we consider all blocs of weight 5 or more in the same class.

Higher Magnitude Error (HME) Expand And Sum is the first decoding function and realises a classic addition over $\mathbb{N}$. Then, the expanded codeword lives in $[\![0, \mathsf{mul}]\!]$. Applied to $\mathbf{y}$, in most cases, the result is in $[\![0, 1]\!]$ but it happens that two errors share the same location modulo 128 in a bloc $(\mathbf{y}'_i)$. This gives an error of magnitude 2. These errors induce a slightly different behavior of the FHT within the same class, affecting the behavior of our Oracle. Fortunately, these higher magnitude errors happen with a low probability. Equation (7) gives the probability of having an error of magnitude at least 2.

$$\begin{aligned}
\mathbb{P}\left(\mathrm{HME}\right) &= \sum_{k=0}^{n_2} \mathbb{P}\left(\mathrm{wt}(\mathbf{y}'_i) = k\right) \times \mathbb{P}\left(\mathrm{HME} \mid \mathrm{wt}(\mathbf{y}'_i) = k\right) \\
&= \sum_{k=0}^{n_2} P_k \times \left(1 - \mathbb{P}\left(\overline{\mathrm{HME}} \mid \mathrm{wt}(\mathbf{y}'_i) = k\right)\right) \\
&= \sum_{k=0}^{n_2} P_k \times \left(1 - \prod_{i=0}^{k} \frac{n_2 - (\mathsf{mul} - 1) \times i}{n_2}\right)
\end{aligned} \tag{7}$$

An HME happens in vector $\mathbf{y}'$ with probabilities $0,53\%$, $0,97\%$ and $0,65\%$ for respectively HQC-128, HQC-192 and HQC-256.

[3] For each line, the sum is not equal to one because of the chosen approximation

3.2 Chosen Ciphertext Attack with Oracle

We use a RM decoding Oracle $\mathcal{O}^{\mathrm{RM}}_{i,b}$ which takes as input an HQC ciphertext $(\mathbf{u}, \mathbf{v})$. $\mathcal{O}^{\mathrm{RM}}_{i,b}$ is able to determine the number of errors corrected by the RM decoder in the ith bloc $\mathbf{y}'_i$ for $i \in [\![0, n_1 - 1]\!]$. Our oracle works in a given range and correctly determine the number of corrected errors if it does not exceed a given threshold τ. The Oracle can be queried for different inputs and returns $b \in [\![0, \tau]\!]$. Notice that in the case of decoding $\mathbf{y}$, the number of decoded errors in a bloc $\mathbf{y}'_i$ is almost always the Hamming weight $\mathrm{wt}(\mathbf{y}'_i)$. We describe how to construct this Oracle $\mathcal{O}^{RM}_{i,b}$ from side-channel leakage in Section 4.

Attack Description Let us focus on a single chosen bloc $\mathbf{y}'_j$, the attack is identical for other blocs of $\mathbf{y}$. In a first step, the Oracle $\mathcal{O}^{RM}_{j,b}$ is queried to known the number of errors to correct in $\mathbf{y}'_j$ which gives a reference value for the next steps. Second, the main idea of the attack is to recursively select $\mathbf{v}_j$ of Hamming weight 1 in order to find a collision with the support of $\mathbf{y}'_j$. Finding a collision implies to modify the number of errors decoded compared to the reference value and then recover an information about $\mathsf{Supp}(\mathbf{y}'_j)$. In fact, for a chosen $\mathbf{v}_j$ there are two cases we can distinguish with the Oracle:

1. $\mathsf{Supp}(\mathbf{y}'_j) \cap \mathsf{Supp}(\mathbf{v}_j) = \mathsf{Supp}(\mathbf{v}_j)$. Then $\mathrm{wt}(\mathbf{v}_j - \mathbf{y}'_j) = \mathrm{wt}(\mathbf{y}'_j) - 1$, the decoder will correct one error less than the reference decoding of $\mathbf{y}'_j$.
$$\mathcal{O}^{\mathrm{RM}}_{j,b}(\mathbf{v} - \mathbf{y}) = O^{\mathrm{RM}}_{j,b}(\mathbf{y}) - 1$$
2. $\mathsf{Supp}(\mathbf{y}'_j) \cap \mathsf{Supp}(\mathbf{v}_j) = \varnothing$. Then $\mathrm{wt}(\mathbf{v}_j - \mathbf{y}_j) = \mathrm{wt}(\mathbf{y}_j) + 1$, the decoder will correct one error more than the reference decoding of $\mathbf{y}$.
$$\mathcal{O}^{\mathrm{RM}}_{j,b}(\mathbf{v} - \mathbf{y}) = O^{\mathrm{RM}}_{j,b}(\mathbf{y}) + 1$$

These observations allow to determine the support of $\mathbf{y}'_j$ by choosing $\mathbf{v}_j$ successively equal to all vectors of Hamming weight 1. By remembering the locations for which the Oracle outputs 1 less than the reference value $O^{\mathrm{RM}}_{j,b}(\mathbf{y})$, we are able to determine the entire support $\mathsf{Supp}(\mathbf{y}'_j)$. Applying this strategy to all blocs of $\mathbf{y}'$ aims at recovering the entire support $\mathsf{Supp}(\mathbf{y}')$.

Divide and Conquer Strategy As described, the attack requires as many queries as the number of bits in $\mathbf{y}'$, i.e. $n_1 n_2$ bits, in order to test all elements of Hamming weight one. However, given that the RM blocs decoding are independent, the attack can be performed in parallel on each bloc. We query the n_1 Oracles $\mathcal{O}^{\mathrm{RM}}_{i,b}$ at the same time, leading to a single query. Then, the minimal number of query needed to recover $\mathbf{y}'$ is reduced to the number of bits in a single bloc, i.e. n_2 bits. As a result, targeting HQC-128 (resp. HQC-192 and HQC-256) requires 384 queries (resp. 640). To know the total number of attack traces needed, this value is multiplied by the number of attack traces necessary to determine a single bit.

4 Building Decoding Oracle with a Side-Channel

In this section, we build a RM decoding Oracle that allows to identify the number of decoded errors. This Oracle is constructed from side-channel leakages and enables to retrieve the secret key $\mathbf{y}'$, as explained in Section 3. We first present our practical set-up which allows traces measurements. Then we describe our Oracle and conduct a leakage assessment with Welch t-test. Finally we evaluate the strength of our Oracle with a different number of training traces and give the cost and performance of the practical attack.

Side-Channel Attack Set-up We realise our measurements on a ARM Cortex M4 micro-controller with a clock frequency of 168 MHz. We record the side-channel leakage from electromagnetic emanations (EM) with a LANGER EMV-Technik near field microprobe ICR HH 100-6. Measurements are registered with a 750M sampling rate oscilloscope ROHDE & SCHWARZ RTO2014. During acquisitions, the communication between the micro-controller and the computer is performed through an UART connection. During the acquisitions, we used an external clock to mitigate the jitter effects among the traces. We extract the Hadamard transform algorithm from the reference implementation of HQC [1] of June 2021. We set a dedicated GPIO pin just before the FHT function to trigger the oscilloscope and reset it after, we will call trace the resulting EM measurement.

4.1 Building the Oracle

We build the Oracle according to the 6 main classes identified with the support probability distribution (see Figure 2). Each of these classes corresponds to a different Hamming weight for $\mathbf{y}'_i$ a bloc of $\mathbf{y}$. Each element of class k is created by randomly sampling its supports, corresponding to k random locations for the ones. Then, among these classes, the proportion of HME vectors is the same as in a HQC instance, following Equation (7). The randomness is provided by the random generator of the micro-controller. We acquired a set of 10.000 traces per class used to evaluate the Oracle. These acquisitions were performed in a random order.

Leakage Assesment We use a Welch t-test to conduct a leakage assessment for the Oracle. The t-value between two sets S_0 and S_1 with their respective cardinality n_0 and n_1, mean of μ_0 and μ_1 and variance of σ_0 and σ_1 is computed with the formula from Equation (8). Usually, a threshold $|t| = 4.5$ is defined, admitting a significant statistical difference with a high degree of confidence when this threshold is exceeded.

$$t = \frac{\mu_0 - \mu_1}{\sqrt{\left(\frac{\sigma_0^2}{n_0} + \frac{\sigma_1^2}{n_1}\right)}} \tag{8}$$

We compute the t-values for each class pair in order to characterize the distinction between them. Results are presented in Figure 4. For each sub-figure, we observe the 7 occurrences of the for loop during the FHT (see Algorithm 7). This test indicates a good level of distinguishability, which could allow to build a classifier.

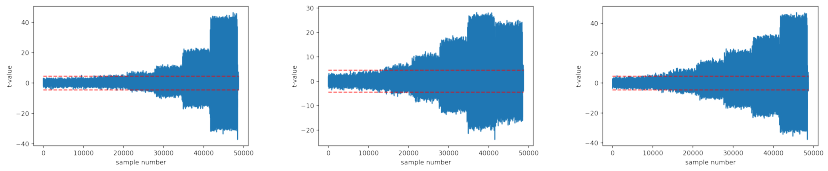
Regions of Interest In the way of the Points of Interest (PoI) selection, the leakage assessment allows us to select Regions of Interest (RoI). This selection allows to keep only relevant parts of the traces and to reduce the computation time of our classifier. As we can see, among the seven occurrences of the for loop, the last one (approx. samples $41,000$ to $48,000$) seems to be very informative (see Figures 4a, 4c, 4f, 4h, 4j, 4l, 4m and 4o). This part of the traces is the first considered RoI for further studies.

However, this RoI does not seem sufficient to build a complete distinguisher, indeed, some results show no leakage in the last occurrence (see Figures 4e, 4g, 4i and 4n). In practice, using only trace segments in this RoI is not enough to mount an attack with a sufficient accuracy. To fill this information gap, we also use another RoI. In order to create a complete distinguisher between every classes, we also extract a RoI from the fifth occurrence (approx. samples $27,300$ to $34,200$). This second RoI allows to distinguish between cases not covered by the first one (see Figures 4e, 4g, 4i, 4k and 4n). For both of these RoI, we keep only the first thousand samples which significantly reduces the computation time and the memory requirements.

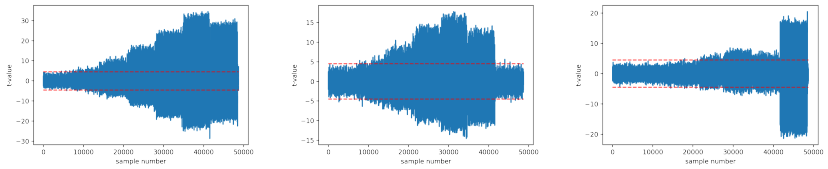
4.2 Results

We build a distinguisher with a Linear Discriminant Analysis (LDA) which is a linear classifier. We use the LDA version from the sklearn python library [18]. We carry out several times the attack with a different number of training traces per class, respectively $1,000$, $2,500$, $5,000$, $10,000$, $20,000$ and $40,000$ traces. The traces are sliced according to the area of interest identified in Section 4.1 and evenly distributed among the target classes.

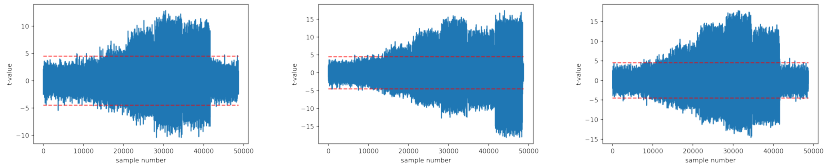
Theoretically, with a perfect Oracle, recovering a single bit in a given bloc requires only one trace. Practically, we quickly see that a single attack trace is not enough to obtain a sufficient success rate (see Figure 5). Then, we build an attack with s traces in order to increase the success rate. Each trace is independently handled by the Oracle and we reconcile the s query outputs with a soft-max technique. Given $(p_1, \cdots, p_\tau) = \sum_{i=1}^{s} (p_{1,i}, \cdots, p_{\tau,i})$ the sum of the probabilities returning by the s instances of the attack for each of the τ classes. The output of the Oracle is given by $b = \mathsf{argmax}(p_1, \cdots, p_\tau)$. We realize the attack several times with s from 0 to 60, we plot in Figure 5 the results of these tests.



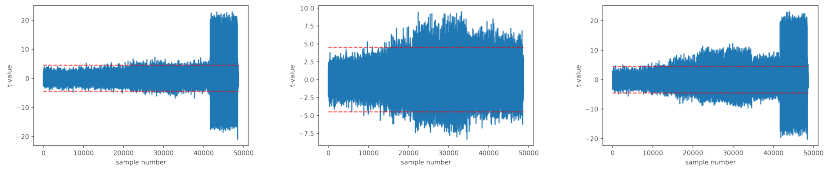
(a) Classes 0 and 1 (b) Classes 0 and 2 (c) Classes 0 and 3



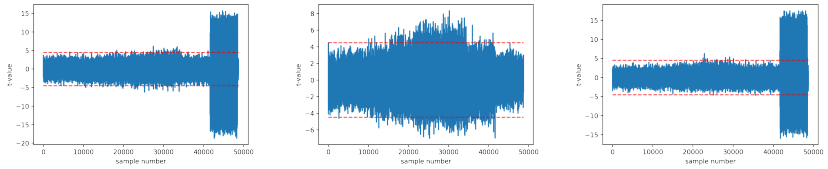
(d) Classes 0 and 4 (e) Classes 0 and 5 (f) Classes 1 and 2



(g) Classes 1 and 3 (h) Classes 1 and 4 (i) Classes 1 and 5



(j) Classes 2 and 3 (k) Classes 2 and 4 (l) Classes 2 and 5



(m) Classes 3 and 4 (n) Classes 3 and 5 (o) Classes 4 and 5

Fig. 4: Welch t-test results between each classes using $10,000$ traces of randomly sampled inputs within the classe. The trace correspond to the FHT. $-4,5$ and $4,5$ are plotted with dashed red lines.

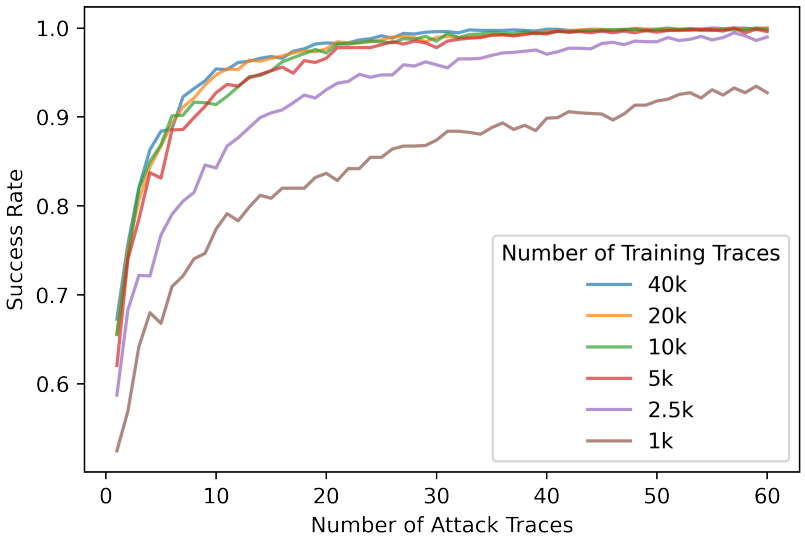


Fig. 5: Single bit success rate recovery depending on the number of attack traces s. Comparison of the success rate between attacks with a different number of training traces per class.

Attack Success Rate and Cost Experiments of figure 5 show that $s = 50$ attack traces are enough to reach a perfect success rate on a bit with a training set of $40,000$ traces per class. We build the attack on our set of measurements with $40,000$ training traces per class, $240,000$ training traces in total. With this set of parameters and our specific training and attack set-up, we are able to recover all bits of $\mathbf{y}'$ with $s \times n_2 = 50 \times 384 = 19,200$ attack traces for HQC-128 (see Section 3.2).

5 Countermeasure

A direct countermeasure against RM decoding distinguisher is the used of a mask. The idea is to hide the sensitive data by dividing its knowledge in several part. Indeed, if the input $\mathbf{c}$ of the FHT satisfies the relation 9,

$$\mathbf{c} = \sum_{i=0}^{n} \mathbf{c}_i \tag{9}$$

by the linearity of the Hadamard transform, the result is given by Equation (10).

$$\mathsf{FHT}(\mathbf{c}) = \sum_{i=0}^{n} \mathsf{FHT}(\mathbf{c}_i) \tag{10}$$

To create a secure masking scheme, the $n-1$ first $\mathbf{c}_i$ must be sampled uniformly at random and the last element $\mathbf{c}_n$ is chosen to satisfy the relation (9). The n order mask countermeasure requires to compute $n+1$ times the FHT. We give the first order masking Hadamard transform in Algorithm 9.

Algorithm 9 Hadamard Transform with first order mask

Input: expanded codeword $\mathbf{c}$ and the multiplicity mul.
Output: expanded codeword transformed structure $\mathbf{c}$

1: $\mathbf{c}_0 \xleftarrow{\$}$ expanded codeword
2: $\mathbf{c}_1 = c - c_0$
3: $\mathbf{c}_0 = \mathsf{FHT}(\mathbf{c}_0)$
4: $\mathbf{c}_1 = \mathsf{FHT}(\mathbf{c}_1)$
5: $\mathbf{c} = \mathbf{c}_0 + \mathbf{c}_1$
6: **return** $\mathbf{c}$

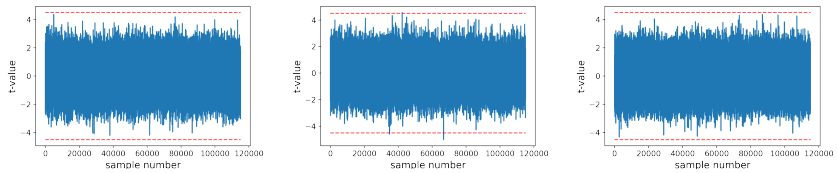
Countermeasure Evaluation An attacker who would like to target the masked version of the Hadamard transform would have to target the n shares in order to retrieve the whole information. Our attack scenario cannot be applied directly against the shares given that the expanded codewords $\mathbf{c}_i$ are randomly sampled. This implies that the shares do not respect the Hamming weight restrictions imposed by our Oracle.

In spite of this, we evaluate the strength of the counter-measure by repeating the experiment as in the Section 4. We compute the FHT with the first order mask (see Algorithm 9) on $60,000$ expanded codewords evenly distributed among the classes. Then, we compute the t-values for each class peer, leading to the results presented in Figure 6. We assume that the significant reduction in the number of observed statistical differences with the Welch t-test ensures that the countermeasure is effective against our attack.

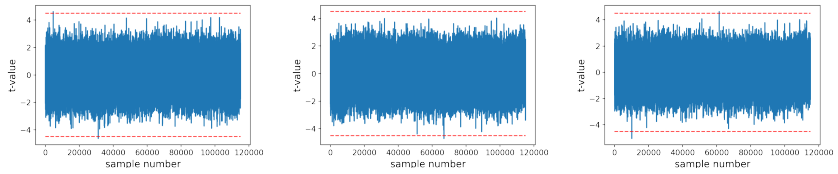
6 Conclusion and Future Work

In this paper we present a new side-channel attack on the RMRS version of HQC which aims at recovering a static secret key. We show that by choosing a certain ciphertext, a part of the secret key is given as input of the decoding algorithm. We build a chosen ciphertext attack from this point by slightly modifying the ciphertext in order to find collision with the secret key. In the paper, we show a strategy, a query sequence, that allows to find collisions and then recover the entire secret key.

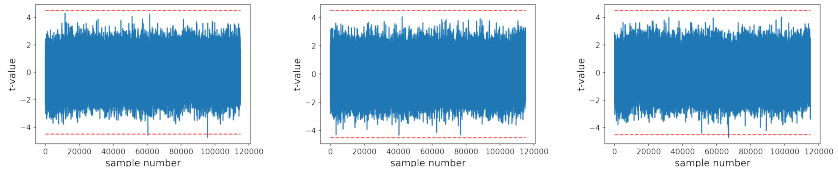
Our attack is based on a side-channel Oracle that is able to distinguish between several decoding patterns. We evaluate our Oracle with electromagnetic side-channel measurements from our Cortex M4 micro-controller set-up and show



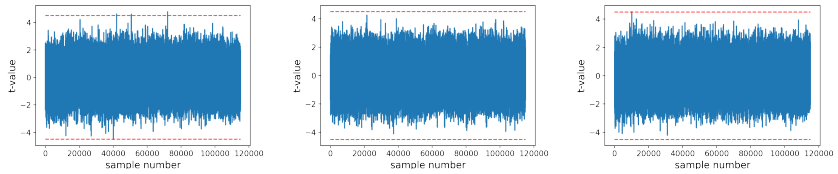
(a) Classes 0 and 1 (b) Classes 0 and 2 (c) Classes 0 and 3



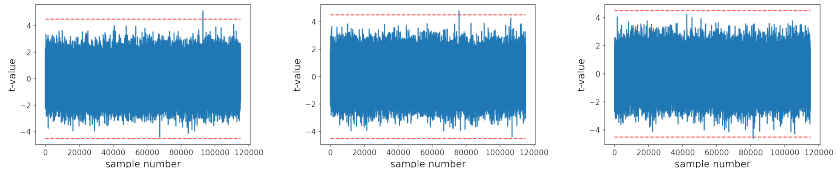
(d) Classes 0 and 4 (e) Classes 0 and 5 (f) Classes 1 and 2



(g) Classes 1 and 3 (h) Classes 1 and 4 (i) Classes 1 and 5



(j) Classes 2 and 3 (k) Classes 2 and 4 (l) Classes 2 and 5



(m) Classes 3 and 4 (n) Classes 3 and 5 (o) Classes 4 and 5

Fig. 6: Welch t-test results between each classes using $10,000$ traces of randomly sampled inputs within the classe. The trace correspond to the FHT with the counter-measure. $-4,5$ and $4,5$ are plotted with dashed red lines.

that it is easy to build and reliable. Indeed, the best performances of our Oracle were observed with a trade-off of $40,000$ training traces per classes, and 50 attack trace to recover a bit on a bloc with a success rate of 1 on our test set measurements. The independence of the decoding among the different blocs allows to parallelize the attack and recover all bits of the secret key with $19,200$ attacks traces.

Our attack is a threat to the security of HQC and contributes to the need of an efficient countermeasure to mitigate such attacks. We propose a simple masking-based countermeasure in order to thwart our attack that doubles the run-time of the target function. As a perspective, the number of attack traces could be reduce by finding more efficient query sequence. For the same purpose, other functions of HQC could play the role of distinguisher, allowing to improve the performance or even to build new attacks.

7 Acknowledgements

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. Aguilar-Melchor, C., Aragon, N., Bettaieb, S., Bidoux, L., Blazy, O., Deneuville, J.C., Gaborit, P., Persichetti, E., Zémor, G.: HQC reference implementation, https://pqc-hqc.org/implementation.html
2. Aguilar-Melchor, C., Aragon, N., Bettaieb, S., Bidoux, L., Blazy, O., Deneuville, J.C., Gaborit, P., Persichetti, E., Zémor, G.: Hamming Quasi-Cyclic (HQC) (2017)
3. Alagic, G., Alperin-Sheriff, J., Apon, D., Cooper, D., Dang, Q., Kelsey, J., Liu, Y.K., Miller, C., Moody, D., Peralta, R., et al.: Status report on the second round of the nist post-quantum cryptography standardization process. US Department of Commerce, NIST (2020)
4. Aragon, N., Gaborit, P., Zémor, G.: HQC-RMRS, an instantiation of the HQC encryption framework with a more efficient auxiliary error-correcting code. arXiv preprint arXiv:2005.10741 (2020)
5. Berlekamp, E., McEliece, R., van Tilborg, H.: On the inherent intractability of certain coding problems (corresp.). IEEE Transactions on Information Theory **24**(3), 384–386 (May 1978). https://doi.org/10.1109/TIT.1978.1055873
6. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: International workshop on cryptographic hardware and embedded systems. pp. 16–29. Springer (2004)
7. Fujisaki, E., Okamoto, T.: Secure integration of asymmetric and symmetric encryption schemes. In: Annual International Cryptology Conference. pp. 537–554. Springer (1999)
8. Fujisaki, E., Okamoto, T.: Secure integration of asymmetric and symmetric encryption schemes. Journal of cryptology **26**(1), 80–101 (2013)

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 (2022)
10. Guo, Q., Hlauschek, C., Johansson, T., Lahr, N., Nilsson, A., Schröder, R.L.: Don't reject this: Key-recovery timing attacks due to rejection-sampling in hqc and bike. Cryptology ePrint Archive (2021)
11. Hofheinz, D., Hövelmanns, K., Kiltz, E.: A modular analysis of the fujisaki-okamoto transformation. In: Kalai, Y., Reyzin, L. (eds.) Theory of Cryptography. pp. 341–371. Springer International Publishing, Cham (2017)
12. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Annual international cryptology conference. pp. 388–397. Springer (1999)
13. Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: Annual International Cryptology Conference. pp. 104–113. Springer (1996)
14. MacWilliams, F.J., Sloane, N.J.A.: The theory of error correcting codes, vol. 16. Elsevier (1977)
15. McEliece, R.J.: A public-key cryptosystem based on algebraic. Coding Thv **4244**, 114–116 (1978)
16. NIST: Submission requirements and evaluation criteria for the post-quantum cryptography standardization process. . (2016)
17. Paiva, T.B., Terada, R.: A timing attack on the HQC encryption scheme. In: International Conference on Selected Areas in Cryptography. pp. 551–573. Springer (2019)
18. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)
19. Petrvalsky, M., Richmond, T., Drutarovsky, M., Cayrel, P.L., Fischer, V.: Differential power analysis attack on the secure bit permutation in the mceliece cryptosystem. In: 2016 26th International Conference Radioelektronika (RADIOELEKTRONIKA). pp. 132–137. IEEE (2016)
20. Rivest, R.L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM **21**(2), 120–126 (1978)
21. 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. Cryptology ePrint Archive (2022)
22. Schamberger, T., Renner, J., Sigl, G., Wachter-Zeh, A.: A power side-channel attack on the CCA2-Secure HQC KEM. In: 19th Smart Card Research and Advanced Application Conference (CARDIS2020) (2020)
23. Schröder, L.: A novel timing side-channel assisted key-recovery attack against HQC. Ph.D. thesis, Wien (2022)
24. Sendrier, N.: Decoding one out of many. In: International Workshop on Post-Quantum Cryptography. pp. 51–67. Springer (2011)
25. Wafo-Tapa, G., Bettaieb, S., Bidoux, L., Gaborit, P., Marcatel, E.: A practicable timing attack against HQC and its countermeasure. Advances in Mathematics of Communications (2020)
26. Walter, C.D.: Sliding windows succumbs to big mac attack. In: International Workshop on Cryptographic Hardware and Embedded Systems. pp. 286–299. Springer (2001)