A Power Side-Channel Attack on the Reed-Muller Reed-Solomon Version of the HQC Cryptosystem

Thomas Schamberger, Lukas Holzbaur, Julian Renner, Antonia Wachter-Zeh, and Georg Sigl

Technical University of Munich, Munich, Germany
{t.schamberger,lukas.holzbaur,julian.renner,antonia.wachter-zeh,sigl}@tum.de

Abstract. The code-based post-quantum algorithm Hamming Quasi-Cyclic (HQC) is a third round alternative candidate in the NIST standardization project. For their third round version the authors utilize a new combination of error correcting codes, namely a combination of a Reed-Muller and a Reed-Solomon code, which requires an adaption of published attacks. We identify that the power side-channel attack by Uneo et al. from CHES 2021 does not work in practice as they miss the fact that the implemented Reed-Muller decoder does not have a fixed decoding boundary. In this work we provide a novel attack strategy that again allows for a successful attack. Our attack does not rely on simulation to verify it success but is proven with high probability for the HQC parameter sets. In contrast to the timing side-channel attack by Guo et al. we are able to reduce the required attack queries by a factor of 12 and are able to eliminate the inherent uncertainty of their used timing oracle. We show practical attack results utilizing a power side-channel of the used Reed-Solomon decoder on an ARM Cortex-M4 microcontroller. In addition, we provide a discussion on how or whether our attack strategy to be usable with the side-channel targets of mentioned related work. Finally, we use information set decoding to evaluate the remaining attack complexity for partially retrieved secret keys. This work again emphasizes the need for a side-channel secure implementation of all relevant building blocks of HQC.

Keywords: Error Correction · HQC · Post-Quantum Cryptography · Power Analysis · Side-Channel Analysis

1 Introduction

The post-quantum cryptography (PQC) contest of NIST is currently in its third round with the goal of reaching standardization of first algorithms in 2022 and a broader portfolio of alternatives in 2024. While the first two rounds were dominated with research on possible performance improvements and optimized implementations the focus of the third round is mainly on the side-channel security of systems, which was also encouraged by NIST in is final report of the second

round [10]. This paper discusses a new side-channel attack against the code-based post-quantum cryptosystem Hamming Quasi Cyclic (HQC) [8], which is an alternative candidate for standardization in the third round of the NIST contest. HQC makes use of the concatenation of two error correction codes in order to offer a good trade-off between error-correction capability and fast encryption and decryption. The authors changed the used codes in their third round submission from a repetition in combination with a BCH code to a concatenated Reed-Muller and Reed-Solomon code. As most works on side-channel attacks against HQC are based on the former combination, these attacks have to be revised.

Related Work Attacks on the second-round version of HQC are mostly based on the observation that a side-channel leakage of the used BCH code can be used to construct a decoding oracle that allows to distinguish whether an error has to be corrected by the BCH decoder during decryption. There are timing side-channel attacks [11,16] against a non-constant time implementation of the BCH decoder as well as a power side-channel attack [13].

Attacks against the current version of the HQC cryptosystem, with a combination of a Reed-Muller and Reed-Solomon code, use parts of the implemented variant of the Fujisaki-Okamoto transformation to build a plaintext-checking oracle. This allows distinguishing if crafted ciphertexts decrypt to the same plaintext dependent on the secret key, again resulting in a possible attack on the whole key using multiple queries to the oracle. Xagawa et al. [17] use a fault injection to skip the ciphertext comparison setup of the transformation resulting in direct access to the plaintext, while Ueno et al. [15] attack the used pseudorandom number generator (PRG) required in the transformation through a power side-channel. Both use an adaption of an attack described in [2,5] to the third round version of HQC. Most recent Guo et al. [3] observed that the implemented fixed-weight sampler in combination with the pseudorandom function (PRF) provide a timing side-channel that can be used as a plaintext checking oracle. They claim a success rate of 87% in retrieving the whole key.

Contributions With their choice of Reed-Muller codes in their third round version of the HQC algorithm the authors implicity induce a different form of decoder, as for Reed-Muller codes the decoder is usually implemented as a maximum likelihood decoder. This decoder class, in contrast to a bounded-distance decoder, as used for the former repetition code, does not have a distinct decoding boundary. In other words the amount of correctable errors is dependent on the support of the error, which in some cases allows correcting more errors than the specified error correction capability of the code. This characteristic breaks the attack attempts by [2,5], which has been unfortunately missed in [15,17] and therefore described attacks do not work. In this paper we provide an attack strategy that again allows successful power side-channel attacks on the third round version of HQC. Our contributions are as follows:

– We show through simulation and with the use of a counterexample that attacks described in [15,17] are not successful. Thus, we develop a new attack

strategy that again allows the retrieval of the secret key through a power side-channel attack.
- While the previous attacks had to rely on simulations or an attack on a few keys to verify the success of their attack strategy, we are able to prove sufficient conditions for the success of our approach. These conditions are fulfilled with very high probability for the parameters of HQC.
- We provide an evaluation of the remaining attack complexity for partially retrieved secret keys with the use of information set decoding.
- Finally, we discuss possible side-channel oracles and show an analysis of the power side-channel from [13] for the updated HQC version.

2 Preliminaries

2.1 Notation

Let $\mathbb{F}_q$ be the finite field of size q. Denote by $\mathbb{F}_2[x_1, \ldots, x_m] = \mathbb{F}_2[\boldsymbol{x}]$ the ring of polynomials in $\boldsymbol{x} = (x_1, \ldots, x_m)$ with coefficients in $\mathbb{F}_2$. For a polynomial $f(\boldsymbol{x}) \in \mathbb{F}_2[\boldsymbol{x}]$ and integer $s \geq 1$, define the evaluation map

$$\begin{aligned} \mathrm{ev}^{\times s} : \quad & \mathbb{F}_2[\boldsymbol{x}] \mapsto \mathbb{F}_2^{s2^m} \\ & \mathrm{ev}^{\times s}(f(\boldsymbol{x})) \mapsto (\underbrace{f(\boldsymbol{a}), f(\boldsymbol{a}), \ldots, f(\boldsymbol{a})}_{s \text{ times}})_{\boldsymbol{a} \in \mathbb{F}_2^m} . \end{aligned}$$

Note that if the degree of each variable $x_1, \ldots, x_m$ is restricted to be at most 1, this is a bijective mapping, i.e., any vector in $\mathbb{F}_2^{2^m}$ can be uniquely associated to a polynomial $f(\boldsymbol{x}) \in \mathbb{F}_2[\boldsymbol{x}]$. For any ordering of the elements $\boldsymbol{a} \in \mathbb{F}_2^{2^m}$ we define the mapping $\chi(\boldsymbol{a})$ from $\mathbb{F}_2^m$ to the indices of the s positions in $\mathrm{ev}^{\times s}(f(\boldsymbol{x}))$ corresponding to $f(\boldsymbol{a})$. For ease of notation, we neglect the dependence of $f(\boldsymbol{x})$ on $\boldsymbol{x}$, if clear from context. By slight abuse of notation, we define the Hamming weight $\mathrm{HW}^{\times s}(f) \coloneqq \mathrm{HW}^{\times s}(\mathrm{ev}^{\times s}(f))$ and Hamming distance $d^{\times s}(f, f') \coloneqq d(\mathrm{ev}^{\times s}(f), \mathrm{ev}^{\times s}(f'))$ of polynomials via the respective evaluation map. The support of a polynomial is defined to be $\mathrm{supp}^{\times s}(f) \coloneqq \{\chi(\boldsymbol{a}) \mid f(\boldsymbol{a}) \neq 0, \boldsymbol{a} \in \mathbb{F}_2^m\}$. If $s = 1$ we omit the $\times 1$ superscript from our notation.

We define $\mathbb{F}_2^{m \times n}$ to be the set of all $m \times n$ matrices over $\mathbb{F}_2$, $\mathbb{F}_2^n = \mathbb{F}_2^{1 \times n}$ for the set of all row vectors of length n over $\mathbb{F}_2$, and define the set of integers $[a, b] \coloneqq \{i : a \leq i \leq b\}$. We index rows and columns of $m \times n$ matrices by $0, \ldots, m-1$ and $0, \ldots, n-1$, where the entry in the i-th row and j-th column of the matrix $\boldsymbol{A}$ is denoted by $A_{i,j}$.

The Hamming weight of a vector $\boldsymbol{a}$ is indicated by $\mathrm{HW}(\boldsymbol{a})$ and the Hamming support of $\boldsymbol{a}$ is denoted by $\mathrm{supp}(\boldsymbol{a}) \coloneqq \{i \in \mathbb{Z} : a_i \neq 0\}$. Let $\mathcal{V}$ be a vector space of dimension n over $\mathbb{F}_2$. We define the product of $\boldsymbol{u}, \boldsymbol{v} \in \mathcal{V}$ as $\boldsymbol{uv} = \boldsymbol{u}\,\mathrm{rot}(\boldsymbol{v})^\top = \boldsymbol{v}\,\mathrm{rot}(\boldsymbol{u})^\top = \boldsymbol{vu}$, where

$$\mathrm{rot}(\boldsymbol{v}) \coloneqq \begin{bmatrix} v_0 & v_{n-1} & \ldots & v_1 \\ v_1 & v_0 & \ldots & v_2 \\ \vdots & \vdots & \ddots & \vdots \\ v_{n-1} & v_{n-2} & \ldots & v_0 \end{bmatrix} \in \mathbb{F}_2^{n \times n}.$$

As a consequence of this definition, elements of $\mathcal{V}$ can be interpreted as polynomials in the ring $\mathcal{R} := \mathbb{F}_2[X]/(X^n - 1)$.

2.2 HQC

The HQC scheme consists of a public code $\mathcal{C} \subseteq \mathbb{F}_2^n$ of length n and dimension k, where it is assumed that both an efficient encoding algorithm **Encode** and an efficient decoding algorithm **Decode** are known publicly. In the following we describe the HQC algorithm as submitted to the third round of the NIST post-quantum standardization contest with its specification last updated in June 2021. We start by introducing the PKE version of the algorithm as shown in Algorithms 1 to 3. Within these algorithms several polynomials are uniformly sampled from $\mathcal{R}$, denoted as $\xleftarrow{\$}$ with the optional argument of specifying the Hamming weight w of the polynomial as well as the randomness θ used to initialize the sampler. The parameter sets of HQC are shown in Table 1.

Algorithm 1: KeyGen

Input: param
Output: pk, sk

$\boldsymbol{h} \xleftarrow{\$} \mathcal{R}$
$(\boldsymbol{x}, \boldsymbol{y}) \xleftarrow{\$(w)} \mathcal{R}^2$
$\boldsymbol{s} \leftarrow \boldsymbol{x} + \boldsymbol{h}\boldsymbol{y}$
$\mathsf{pk} = (\boldsymbol{h}, \boldsymbol{s})$
$\mathsf{sk} = (\boldsymbol{x}, \boldsymbol{y})$
return pk, sk

Algorithm 2: Encrypt

Input: $\boldsymbol{m}, \mathsf{pk}, \theta$
Output: c

$\boldsymbol{e}' \xleftarrow{\$(w_\mathrm{e}, \theta)} \mathcal{R}$
$(\boldsymbol{r}_1, \boldsymbol{r}_2) \xleftarrow{\$(w_\mathrm{r}, \theta)} \mathcal{R}^2$
$\boldsymbol{u} \leftarrow \boldsymbol{r}_1 + \boldsymbol{h}\boldsymbol{r}_2$
$\boldsymbol{v} \leftarrow \mathbf{Encode}(\boldsymbol{m}) + \boldsymbol{s}\boldsymbol{r}_2 + \boldsymbol{e}'$
return $c = (\boldsymbol{u}, \boldsymbol{v})$

Algorithm 3: Decrypt

Input: $\mathsf{sk} = (\boldsymbol{x}, \boldsymbol{y})$
$c = (\boldsymbol{u}, \boldsymbol{v})$
Output: $\boldsymbol{m}$

$\boldsymbol{v}' \leftarrow \boldsymbol{v} - \boldsymbol{u}\boldsymbol{y}$
$\boldsymbol{m} \leftarrow \mathbf{Decode}(\boldsymbol{v}')$
return $\boldsymbol{m}$

	shortened RS code $[n_1, k, d_{RS}]$	duplicated RM code $[n_2, k_{RM}, d_{RM}, s]$	n	w	$w_\mathrm{r} = w_\mathrm{e}$
HQC-128	[46, 16, 31]	[384, 8, 192, 3]	17,669	66	75
HQC-192	[56, 24, 33]	[640, 8, 320, 5]	35,851	100	114
HQC-256	[90, 32, 49]	[640, 8, 320, 5]	57,637	131	149

Table 1: Parameter sets of HQC [9]

The HQC authors use the PKE version of HQC to construct an IND-CCA2 secure KEM. Using this KEM, a random shared secret K can be exchanged where the sender applies encapsulation (Algorithm 4) and the receiver decapsulation (Algorithm 5). These algorithms use three different hash functions $\mathcal{G}$,$\mathcal{H}$, and $\mathcal{K}$, which are based on SHAKE256 with 512 bits of output. In order to counteract chosen-chipertext attacks, the decrypted message is re-encrypted and compared to the original ciphertext input. Only if both ciphertexts are equal K gets released otherwise the decapsulation is aborted. In order for the re-encryption to be possible the sampling of the random elements has to be deterministic, which is ensured by deriving a seed from the message which is used to initialize the sampler.

Algorithm 4: Encapsulate

Input: pk
Output: $K, c = (\boldsymbol{u}, \boldsymbol{v}), \boldsymbol{d}$

$\boldsymbol{m} \xleftarrow{\$} \mathbb{F}_2^k$
$\theta \leftarrow \mathcal{G}(\boldsymbol{m})$
$c \leftarrow \mathsf{Encrypt}(\mathsf{pk}, \boldsymbol{m}, \theta)$
$K \leftarrow \mathcal{K}(\boldsymbol{m}, c)$
$\boldsymbol{d} \leftarrow \mathcal{H}(\boldsymbol{m})$

Algorithm 5: Decapsulate

Input: $\mathsf{sk}, c, \boldsymbol{d}$
Output: K

$\boldsymbol{m}' \leftarrow \mathsf{Decrypt}(\mathsf{sk}, c)$
$\theta' \leftarrow \mathcal{G}(\boldsymbol{m}')$
$c' \leftarrow \mathsf{Encrypt}(\mathsf{pk}, \boldsymbol{m}', \theta')$
if $c \neq c'$ *and* $\boldsymbol{d} \neq \mathcal{H}(\boldsymbol{m}')$ **then**
  abort
else
  $K \leftarrow \mathcal{K}(\boldsymbol{m}, c)$

2.3 Choice of Error Correcting Code $\mathcal{C}$

For the third round version of HQC the code $\mathcal{C}$ is instantiated as a concatenated code with a Reed-Muller (RM) code as the inner and a Reed-Solomon (RS) code as the outer code. The function **Encode** describes the encoding of a message $\boldsymbol{m} \in \mathbb{F}_2^k$ into $\tilde{\boldsymbol{m}} \in \mathbb{F}_2^{n_1 n_2}$ using the concatenated code. First the outer RS code is used to encode $\boldsymbol{m}$ into $\boldsymbol{m}_1 \in \mathbb{F}_{2^8}^{n_1}$, followed by encoding each coordinate $\boldsymbol{m}_{1,i}$ of $\boldsymbol{m}_1$ into $\tilde{\boldsymbol{m}}_{1,i} \in \mathbb{F}_2^{n_2}$ using the inner duplicated RM code. The duplicated encoding works in two phases. First, each $\boldsymbol{m}_{1,i}$ is encoded with the underlying [128,8,64]-RM code to obtain $\bar{\boldsymbol{m}}_{1,i}$, which is then duplicated based on the multiplicity s (see Table 1) resulting in $\tilde{\boldsymbol{m}}_{1,i}$. In other words the final encoding result is constructed as $\tilde{\boldsymbol{m}} = (\tilde{\boldsymbol{m}}_{1,0}, \ldots, \tilde{\boldsymbol{m}}_{1,n_1-1}) \in \mathbb{F}_2^{n_1 n_2}$.

The function **Decode** describes the decoding of an input in $\mathbb{F}_2^{n_1 n_2}$ to a message $\boldsymbol{m} \in \mathbb{F}_2^k$. First the individual $\tilde{\boldsymbol{m}}_{1,i}$ are decoded with the duplicated RM decoder ($\mathfrak{D}_{\mathcal{RM}}$), which results in the input to RS decoder ($\mathfrak{D}_{\mathcal{RS}}$) as an element in $\mathbb{F}_{2^8}^{n_1}$. Finally, the RS decoding results in the message $\boldsymbol{m} \in \mathbb{F}_2^k$.

Definition 1 (First Order Reed-Muller Code). *Denote* $\boldsymbol{x} = (x_1, \ldots, x_m)$. *Define the code* $\mathcal{RM}^{\times s}(m)$ *to be*

$$\mathcal{RM}^{\times s}(m) = \left\{ \mathrm{ev}^{\times s}(f(\boldsymbol{x})) \;\middle|\; f(\boldsymbol{x}) \in \mathbb{F}_2[\boldsymbol{x}], \deg(f(\boldsymbol{x})) \leq 1 \right\}.$$

We refer to s *as the multiplicity of the RM code. If* $s = 1$ *we omit the superscript and write* $\mathcal{RM}(m)$.

For details on RM codes and their properties we refer the reader to [7, Chapter 13].

3 Novel Oracle-Based Side-Channel Attack

In this section we describe our chosen-ciphertext attack against the HQC cryptosystem which is able to retrieve the secret key polynomial $\boldsymbol{y}$ during decryption/decapsulation. We start by describing the support distribution of $\boldsymbol{y}$, which is essential for the attack. In a second step we introduce the general idea of our

attack and focus on the characteristics of RM codes that render some published attacks unsuccessful. As a third step we describe our attack strategy based on a close-to-zero oracle. We additionally provide attack results and compare the required oracle queries to related work. Finally, we show how to retrieve the secret key from partial attack results using information set decoding.

3.1 Support Distribution of $\boldsymbol{y}$

With the proposed HQC parameter sets, $\boldsymbol{y}$ is sampled as a sparse polynomial with $\mathrm{HW}(\boldsymbol{y}) = w$. As our attack targets each duplicated RM codeword of length n_2 individually, we split $\boldsymbol{y}$ into corresponding chunks $\boldsymbol{y}_i^{(0)}$ with $0 \leq i \leq n_1$. In order to prevent algebraic attacks, $\boldsymbol{y}$ is chosen to be of length n, which is the smallest primitive prime greater than $n_1 n_2$. Therefore, $\boldsymbol{y}$ contains a second part $\boldsymbol{y}^{(1)}$ consisting of the remaining $n - n_1 n_2$ bits.

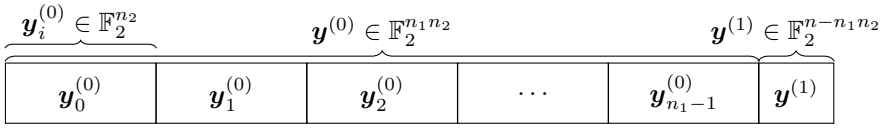

Fig. 1: Different parts of the secret key $\boldsymbol{y} \in \mathbb{F}_2^n$.

Our proof of the attack is defined for a maximum Hamming weight of all $\boldsymbol{y}_i^{(0)}$. For this we define $\boldsymbol{y}_{w,max} = \max\{\mathrm{HW}(\boldsymbol{y}_0^{(0)}), \ldots, \mathrm{HW}(\boldsymbol{y}_{n_1-1}^{(0)})\}$ as the maximum Hamming weight of all chunks of $\boldsymbol{y}^{(0)}$. We determine the probabilities of certain $\boldsymbol{y}_{w,max}$ by simulating the weight distribution of 10 million samples of $\boldsymbol{y}^{(0)}$ with the results shown in Table 2.

$y_{w,max}$	1	2	3	4	5	6	7	8	9
HQC-128	0%	≈0%	3.75%	48.77%	86.49%	97.44%	99.59%	99.94%	99.99%
HQC-192	0%	≈0%	0.01%	10.74%	57.96%	88.50%	97.57%	99.56%	99.93%
HQC-256	0%	≈0%	0.09%	20.83%	71.87%	93.94%	98.96%	99.84%	99.97%

Table 2: Probabilities that $\boldsymbol{y}^{(0)}$ is generated such that the weight of $\boldsymbol{y}_i^{(0)}$ for $i = 0, \ldots, n_1 - 1$ is at most $y_{w,max}$.

With our attack strategy we are able to attack $\boldsymbol{y}^{(0)}$, as only this part acts as an input to the decoder. Nevertheless, we discuss methods to retrieve the whole $\boldsymbol{y}$ in Section 3.2. The probability that $\mathrm{HW}(\boldsymbol{y}^{(1)}) > 0$ can be computed by $1 - \binom{n_1 \cdot n_2}{w} / \binom{n}{w}$. Considering the parameters of HQC-128, HQC-192, and HQC-256 we determine the respective probabilities to be 1.85%, 3.02%, and 8.07%. This means that in most cases it suffices to determine the $\boldsymbol{y}^{(0)}$ because there are no ones in $\boldsymbol{y}^{(1)}$.

3.2 General Attack Idea

With our chosen-ciphertext attack we are able to determine all parts of the secret key $\boldsymbol{y}_i^{(0)}$ individually and in sequential manner. Within the Hamming

Quasi-Cyclic (HQC) algorithm the only operation utilizing the secret key is the decoding of the vector $\boldsymbol{v}' = \boldsymbol{v} - \boldsymbol{u}\boldsymbol{y}$ during decryption (c.f. Algorithm 3). By setting $\boldsymbol{u} = (1, 0 \ldots, 0) \in \mathbb{F}_2^n$ the decoder input results in $\boldsymbol{v}' = \boldsymbol{v} - \boldsymbol{y}$. As $\boldsymbol{v}$ is part of the ciphertext $c = (\boldsymbol{u}, \boldsymbol{v})$, it is controllable by the attacker. By setting it to a valid codeword $\in \mathcal{C}$, $\boldsymbol{y}$ can be seen as an error that has to be corrected by the decoder. Note that due to the sparseness of $\boldsymbol{y}$ it is sufficient to only retrieve its support.

Now the general idea of the attack is to choose $\boldsymbol{v}$ such that the decoding result depends on $\boldsymbol{y}_i^{(0)}$, revealing its support. In the case of HQC an attacker requires access to the individual RM decoder results, as the respective input consists of $\boldsymbol{y}_i^{(0)}$ subtracted from its corresponding part of $\boldsymbol{v}$. It is not possible to directly attack the decapsulation of HQC (Algorithm 5), as it includes a check for the validity of the ciphertext that does not reveal information about $\boldsymbol{m}$ in case of failure. Nevertheless, a side-channel can be used in order to construct an oracle that again reveals information about the decoding result. We therefore construct an oracle that is able to determine whether the RM decoder decoded to the all-zero or a non-zero codeword in a given position. The oracle is formally defined in Definition 2. We discuss different side-channels, which can be used to construct this oracle, in Section 4.

Definition 2 (Close-to-0-Oracle). *Let $\mathcal{C}$ be an $\mathcal{RM}(m)$ code. Define $\mathfrak{D}_0^{\boldsymbol{e}} : \mathbb{F}_q^n \mapsto \{\mathsf{True}, \mathsf{False}\}$ with $\boldsymbol{e} \in \mathbb{F}_q^n$ to be the function given by*

$$\mathfrak{D}_0^{\boldsymbol{e}}(\boldsymbol{r}) = \begin{cases} \mathsf{True}, & \text{if } \mathfrak{D}_{\mathcal{RM}}(\boldsymbol{r} + \boldsymbol{e}) = \boldsymbol{0}, \\ \mathsf{False}, & \text{else,} \end{cases}$$

where $\mathfrak{D}_{\mathcal{RM}}$ denotes a decoder for the RM code.

By querying the oracle and therefore having access to the decoding result our attack strategy as well as the related work is based on two steps. First, an input has to be found (or set for some attacks) that, after the subtraction of the corresponding $\boldsymbol{y}_i^{(0)}$, lies exactly at the decoding boundary of the RM decoder. An example is to find an input that lies exactly one error above the border meaning the input results in a decoding error, i.e. in the decoder not returning the all-zero codeword $\boldsymbol{0}$, and therefore the oracle returns False. This implies that if we set an additional bit in the input, which is in the support of $\boldsymbol{y}_i^{(0)}$, we reduce the error resulting in a successful decoding indicated by the oracle as True. Now in the second attack step we can query the oracle by successively inverting each bit of the input we found in the first step. In this process an oracle result of True, i.e., a successful decoding to the all-zero codeword, indicates that this position is in the support of $\boldsymbol{y}_i^{(0)}$. This allows to retrieve the whole support of the attacked secret key block. By repeating this approach for all n_1 RM blocks we can retrieve the complete $\boldsymbol{y}^{(0)}$.

Limitations of Previous Works With the change of the used codes in the third round version of the HQC, the first attack step, namely finding an input

that lies at the decoding border of the internal code, has to be changed due to the use of a different decoder type. This is the case as the decoder of the now used RM code is implemented as an ML decoder, where ties are resolved in favor of the word of smaller lexicographical order. A ML decoder is formally defined as follows:

Definition 3 (ML Decoder). *Let $\mathcal{C}$ be an $[n,k]_q$ code. Define $\mathfrak{D}_{\mathsf{ML}} : \mathbb{F}_q^n \mapsto \mathcal{C}$ to be a function returning the codeword that maximizes the probability $P(\boldsymbol{r}|\boldsymbol{c})$, i.e.,*

$$\mathfrak{D}_{\mathsf{ML}}(\boldsymbol{r}) = \arg\max_{\boldsymbol{c}\in\mathcal{C}} P(\boldsymbol{r}|\boldsymbol{c}) \ .$$

If this choice is not unique, it returns the word that is smaller in lexicographical order.

In the Hamming metric and without considering soft information an ML decoder translates to a function returning the codeword of the smallest Hamming distance to the given vector, i.e., $\mathfrak{D}_{\mathsf{ML}}(\boldsymbol{r}) = \arg\min_{\boldsymbol{c}\in\mathcal{C}} d(\boldsymbol{r},\boldsymbol{c})$. Note that ML decoding is known to be very complex and therefore rarely used in practice. However, for a few code classes, such as first-order RM codes, efficient decoders are known, a fact that is exploited in this system. Most other systems based on algebraic codes, such as Classic McEliece [1], instead employ bounded-distance decoders, which decode *any* error up to a given weight and fail if no codeword is within this specified radius[1]. On the other hand, for a symmetric memoryless channel an ML decoder *always* returns (one of) the codeword(s) closest to the received word, regardless of its distance to the received word. Importantly, this implies that the behavior of this decoder does not only depend on the *number of errors* but also on the *positions of these errors*. However, this independence of the error positions in a bounded-distance decoder is essential to some known attack strategies such as [2,5]. Hence, while the setup might look similar, these methods cannot be directly applied to a system employing an ML decoder instead of a bounded-distance decoder. For instance, the side-channel attack in [17, Section C.7][2] claims that the method for determining an additive error vector from oracle outputs, given in [2, Figure 7], also applies the third round version of HQC. In Appendix A we show that this leads to incorrect outputs of the algorithm, which are caused by exactly this difference in behavior between an ML and a bounded-distance decoder, rendering their described attack unsuccessful.

Retrieval of $\boldsymbol{y}^{(1)}$ If $\boldsymbol{y}^{(0)}$ has been retrieved completely and error free, we can use the published linear algebraic approach shown in [13] to get the remaining

[1] The previous version of the HQC system employed repetition codes of odd length instead of RM codes. It is well-known that this class of codes is *perfect*, i.e., the unique decoding error balls centered on the codewords fill the entire space. In this specific case, a bounded-distance decoder with radius $(d-1)/2$ is equivalent to an ML decoder. Note that first-order RM codes are *not perfect*, so this special case does not apply here.

[2] Note that the description of the attack in [15] is based on the same assumptions, as it directly refers to [17].

part of the secret key defined as $\boldsymbol{y}^{(1)}$. Assuming that $\mathrm{HW}(\boldsymbol{y}^{(1)}) \leq 2$ the resulting work factor of this approach for HQC-128, HQC-192 and HQC-256 is $2^{19.02}$, $2^{24.08}$ and $2^{30.32}$, respectively. In the case that $\boldsymbol{y}^{(0)}$ is only partially retrieved we have to use information set decoding, as described in Section 3.4, which directly retrieves the complete secret key $\boldsymbol{y}$ from the partial information.

3.3 Description of the Attack Strategy

In this section we introduce our attack strategy that considers the characteristics of the RM ML decoder and therefore again allows for the correct retrieval of $\boldsymbol{y}^{(0)}$. It is based on two algorithms, where first the strategy to find an input word that lies at the decoder border is described as Algorithm 6 and then the strategy to retrieve the support of the error with the use of multiple of these words as Algorithm 7. We start by introducing the reasoning for our strategy leading to a formal proof to successfully retrieve $\boldsymbol{y}^{(0)}$ if the Hamming weight of the respective RM block is smaller than $\frac{d_{\mathcal{RM}}}{4}$. Our simulation of $y_{w,max}$ (see Table 2) indicates that this condition holds for nearly all possible keys of HQC, as 99.9% of simulated keys show a $y_{w,max}$ of 9 with $d_{\mathcal{RM}}$ being 192 for HQC-128 and 320 for HQC-192/HQC-256 (c.f. Table 1), respectively. We conclude this section with a discussion of the required oracle calls of our strategy in comparison to related work.

Algorithm 6: FindWordAtBorder

Input : Oracle function $\mathfrak{D}_0^e$
Sets $\hat{\mathcal{I}}, \check{\mathcal{I}} \subset [0, n_2 - 1]$
Output: Vector $\boldsymbol{r} \in \mathbb{F}_2^{n_2}$

$\boldsymbol{r} \leftarrow \mathbf{0}$
for $\xi \in \hat{\mathcal{I}} \cap \check{\mathcal{I}}$ **do**
  if $\mathfrak{D}_0^e(\boldsymbol{r}) = \mathsf{True}$ **then**
    $r_\xi \leftarrow 1$
  else
    return *Vector* $\boldsymbol{r} \in \mathbb{F}_2^{n_2}$
for $\xi \in \hat{\mathcal{I}} \setminus \check{\mathcal{I}}$ **do**
  if $\mathfrak{D}_0^e(\boldsymbol{r}) = \mathsf{True}$ **then**
    $r_\xi \leftarrow 1$
  else
    return *Vector* $\boldsymbol{r} \in \mathbb{F}_2^{n_2}$

Algorithm 7: FindError

Input : Oracle function $\mathfrak{D}_0^e$
Sets $\mathcal{I}_1, \mathcal{I}_2 \subset [0, n_2 - 1]$
Output: Vector $\tilde{\boldsymbol{e}} \in \mathbb{F}_2^{n_2}$

$\tilde{\boldsymbol{e}} \leftarrow \mathbf{0}$
for $\hat{\mathcal{I}} \in \{\mathcal{I}_1, [0, n_2 - 1] \setminus \mathcal{I}_1\}$ **do**
  for $\check{\mathcal{I}} \in \{\mathcal{I}_2, [0, n_2 - 1] \setminus \mathcal{I}_2\}$ **do**
    $\boldsymbol{r} \leftarrow$ FindWordAtBorder$(\mathfrak{D}_0^e, \hat{\mathcal{I}}, \check{\mathcal{I}})$
    $\hat{\boldsymbol{e}} \leftarrow \boldsymbol{r}$
    for $\xi \in \hat{\mathcal{I}} \cap \check{\mathcal{I}}$ **do**
      $r_\xi \leftarrow r_\xi + 1$
      if $\mathfrak{D}_0^e(\boldsymbol{r}) = \mathsf{True}$ **then**
        $\hat{e}_\xi \leftarrow \hat{e}_\xi + 1$
      $r_\xi \leftarrow r_\xi + 1$
    $\tilde{\boldsymbol{e}}_{\hat{\mathcal{I}} \cap \check{\mathcal{I}}} \leftarrow \hat{\boldsymbol{e}}_{\hat{\mathcal{I}} \cap \check{\mathcal{I}}}$
return *Vector* $\tilde{\boldsymbol{e}} \in \mathbb{F}_2^{n_2}$

To begin, we show some general results on the intersection of the supports of $\mathcal{RM}(m)$ codewords. Note that there exists an extensive literature on RM codes and their supports are well understood. For completeness, we nevertheless

include the following statement in the form required to prove the main results of this section. As the statements and proofs in the following heavily rely on the properties of the multivariate polynomials associated with each RM codeword, we denote all vectors in the following by the polynomial which results in the respective vector when evaluated in $\mathbb{F}_2^{2^m}$.

Lemma 1. *Consider two polynomials* $\hat{p}, \check{p} \in \mathbb{F}_2[\boldsymbol{x}]$ *with* $\deg(\hat{p}) = \deg(\check{p}) = 1$ *and* $\check{p} \notin \{\hat{p}, \hat{p}+1\}$. *Denote* $d = 2^{m-1}$. *Then, for any* $f \in \mathcal{RM}(m)$ *we have*

$$|\operatorname{supp}(f) \cap \operatorname{supp}(\hat{p}\check{p})| = \mathrm{HW}(f\hat{p}\check{p}) = \begin{cases} 0, & \text{if } f \in \{0, \hat{p}+1, \check{p}+1, \hat{p}+\check{p}\} \\ \frac{d}{2}, & \text{if } f \in \{1, \hat{p}, \check{p}, \hat{p}+\check{p}+1\} \\ \frac{d}{4}, & \text{else.} \end{cases}$$

Proof. The first case follows from observing that $f\hat{p}\check{p} = 0$ for these polynomials[3]. It is well-known that any codeword $p \in \mathcal{RM}(m)$, except the all-zero and the all-one word, i.e., any word with $\deg(p) = 1$, is of weight $d = 2^{m-1}$.Since $\deg(\hat{p}) = \deg(\check{p}) = 1$ and $\check{p} \notin \{\hat{p}, \hat{p}+1\}$ we have $\deg(\hat{p}+\check{p}) = 1$. Therefore, $\mathrm{HW}(\hat{p}+\check{p}) = d$ and we get

$$\begin{aligned} \mathrm{HW}(\hat{p}+\check{p}) &= \mathrm{HW}(\hat{p}) + \mathrm{HW}(\check{p}) - 2\mathrm{HW}(\hat{p}\check{p}) \\ d &= 2d - 2\mathrm{HW}(\hat{p}\check{p}) \\ \frac{d}{2} &= \mathrm{HW}(\hat{p}\check{p}) \ . \end{aligned}$$

The second case follows since we have $\operatorname{supp}(\hat{p}\check{p}) \subset \operatorname{supp}(f)$ for any $f \in \{1, \hat{p}, \check{p}, \hat{p}+\check{p}+1\}$. Now consider some $f \in \mathcal{RM}(m) \setminus \{0, \hat{p}+1, \check{p}+1, \hat{p}+\check{p}, 1, \hat{p}, \check{p}, \hat{p}+\check{p}+1\}$ and note that $\deg(f) = 1$. Observe that the supports of the polynomials $\{\hat{p}\check{p}, \hat{p}(\check{p}+1), (\hat{p}+1)\check{p}, (\hat{p}+1)(\check{p}+1)\}$ partition the 2^m codeword positions. Hence, by the pigeonhole principle, there exists some $\bar{p} \in \{\hat{p}(\check{p}+1), (\hat{p}+1)\check{p}, (\hat{p}+1)(\check{p}+1)\}$ with

$$\mathrm{HW}(\bar{p}f) \geq \left\lceil \frac{\mathrm{HW}(f) - \mathrm{HW}(\hat{p}\check{p}f)}{3} \right\rceil \geq \frac{d - \mathrm{HW}(\hat{p}\check{p}f)}{3} \ .$$

Further, it is easy to check that $\hat{p}\check{p}+\bar{p} \in \{\hat{p}, \check{p}, \hat{p}+\check{p}+1\}$, which implies $\deg(\hat{p}\check{p}+\bar{p}) = 1$ and $\mathrm{HW}(\hat{p}\check{p}+\bar{p}) = d$. Now, towards a contradiction, assume $\mathrm{HW}(f\hat{p}\check{p}) > \frac{d}{4}$. Then, we have

$$\begin{aligned} d(\hat{p}\check{p}+\bar{p}, f) &= \mathrm{HW}(\hat{p}\check{p}+\bar{p}) + \mathrm{HW}(f) - 2\mathrm{HW}((\hat{p}\check{p}+\bar{p})f) \\ &= 2d - 2(\mathrm{HW}(\hat{p}\check{p}f) + \mathrm{HW}(\bar{p}f)) \\ &\leq 2\left(d - \left(\mathrm{HW}(\hat{p}\check{p}f) + \frac{d - \mathrm{HW}(\hat{p}\check{p}f)}{3}\right)\right) \\ &\leq 2\Big(d - \frac{d + 2\mathrm{HW}(\hat{p}\check{p}f)}{3}\Big) \end{aligned}$$

[3] Note that $f^2 = f$ in $\mathbb{F}_2[x]$, so $(\hat{p}+1)\hat{p}\check{p} = \hat{p}^2\check{p} + \hat{p}\check{p} = 2\hat{p}\check{p} = 0$.

$$< 2\Big(d - \frac{d + 2\frac{d}{4}}{3}\Big) = d \ .$$

As both $\hat{p}\check{p}+\bar{p}$ and f are in $\mathcal{RM}(m)$, this can only be true if $\hat{p}\check{p}+\bar{p} = f$. However, we have $\hat{p}\check{p}+\bar{p} \in \{\hat{p}, \check{p}, \hat{p}+\check{p}+1\}$ and therefore, by definition of f, a contradiction. Now assume there exists an $f' \in \mathcal{RM}(m) \setminus \{0, \hat{p}+1, \check{p}+1, \hat{p}+\check{p}, 1, \hat{p}, \check{p}, \hat{p}+\check{p}+1\}$ with $\mathrm{HW}(f\hat{p}\check{p}) < \frac{d}{4}$ and note that this set is closed under inversions, i.e., also contains $f'+1$. Then, we have $\mathrm{HW}((f'+1)\hat{p}\check{p}) > \frac{d}{4}$, which cannot be true, as shown above. □

Using these results, we now show that the output of Algorithm 6 always results in a word that causes a specific ML decoding result, under certain non-restrictive assumptions.

Lemma 2. *Denote by $\hat{p}, \check{p} \in \mathbb{F}_2[\boldsymbol{x}]$ two polynomials with $\deg(\hat{p}) = \deg(\check{p}) = 1$ and $\check{p} \notin \{\hat{p}, \hat{p}+1\}$. Then, for $r = \mathsf{FindWordAtBorder}(\mathfrak{D}_0^{\boldsymbol{e}}, \mathrm{supp}(\hat{p}), \mathrm{supp}(\check{p}))$ as in Algorithm 6 it holds that $\mathfrak{D}_{\mathcal{RM}}(r+e) \in \{\hat{p}, \check{p}, \hat{p}+\check{p}+1\}$ and the decision is not the result of a tie in the distance with some other word $\mathcal{RM}(m) \setminus \mathcal{F}$.*

Proof. Denote $\mathcal{F} = \{\hat{p}, \check{p}, \hat{p}+\check{p}+1\}$. First note that the algorithm always returns a word r such that $\mathfrak{D}_0^{\boldsymbol{e}}(r) = \mathsf{False}$. Clearly, this statement would only be false if $\mathfrak{D}_0^{\boldsymbol{e}}(r) = \mathsf{True}$ for all steps in the for loops of Lines 2 and 7. To see that this cannot be the case, consider the $\frac{d}{4}$-th iteration in the for-loop of Line 7. In this iteration we have $\mathrm{HW}(r) = \frac{3}{4}d$ and $\mathrm{supp}(r) \subset \hat{\mathcal{I}} = \mathrm{supp}(\hat{p})$, where $\hat{p} \in \mathcal{RM}(m)$ by definition. It follows that $r+e$ is in the unique decoding ball of $\hat{p}$, since

$$\begin{aligned} d(r+e, \hat{p}) &= \mathrm{HW}(\hat{p}+r+e) \\ &\leq \mathrm{HW}(\hat{p}+r) + \mathrm{HW}(e) \\ &= d - \mathrm{HW}(r) + \mathrm{HW}(e) < \frac{d}{2} \ . \end{aligned}$$

In this case, an ML decoder for the RM code would decide for $\hat{p}$, and it holds that $\mathfrak{D}_0^{\boldsymbol{e}}(r) = \mathsf{False}$ and $\mathfrak{D}_{\mathcal{RM}}(r+e) = \hat{p} \in \{\hat{p}, \check{p}, \hat{p}+\check{p}+1\}$. Note that this also implies $\mathrm{HW}(\hat{p}(\check{p}+1)r) \leq \frac{d}{4}$ for any returned word r. Now consider the case that Algorithm 6 terminates in the for loop of Line 2, i.e., for an r with $\mathrm{supp}(r) \subseteq (\hat{\mathcal{I}} \cap \check{\mathcal{I}})$. For this case, we show a statement that is slightly stronger than required, namely, we prove that for any $f \in \mathcal{RM}(m) \setminus (\mathcal{F} \cup \{0\})$ we have $d(r+e, 0) < d(r+e, f)$, which implies that f cannot be the outcome of an ML decoder[4]. To begin, observe that $\mathfrak{D}_{\mathcal{RM}}(r+e) \neq 1$ since $\mathrm{HW}(r+e) \leq \mathrm{HW}(r)+\mathrm{HW}(e) < \frac{n}{4}+\frac{d}{4}$ and therefore $\boldsymbol{d}(r+e, 0) = \mathrm{HW}(r+e) < n - \mathrm{HW}(r+e) = d(r+e, 1)$, so the ML decoder does not decode to the all-one word in this case. If $\mathrm{HW}(r) \leq \frac{d}{4}$, we get $\mathrm{HW}(r+e) < \frac{d}{2}$ and an ML decoder always decides for 0, i.e., $\mathfrak{D}_0^{\boldsymbol{e}} = \mathsf{True}$, so we can assume that $\mathrm{HW}(r) > \frac{d}{4}$ when Algorithm 6 terminates. Denote $\bar{\mathcal{F}} = \mathcal{RM}(m) \setminus (\mathcal{F} \cup \{0, 1\})$. Now consider some $f \in \bar{\mathcal{F}}$ and note that

[4] Note that this does *not* imply that the outcome is 0, since one of the words of $\mathcal{F}$ could still be closer to $r+e$ than 0.

$\mathrm{supp}(\hat{p}\check{p}) = \hat{\mathcal{I}} \cap \check{\mathcal{I}}$. Then, we have

$$\begin{aligned} d(r+e,f) &= \mathrm{HW}(f+r+e) \\ &\geq \mathrm{HW}(f+r) - \mathrm{HW}(e) \\ &= \mathrm{HW}(\hat{p}\check{p}(f+r)) + \mathrm{HW}((\hat{p}\check{p}+1)(f+r)) - \mathrm{HW}(e) \\ &= \mathrm{HW}(\hat{p}\check{p}(f+r)) + \mathrm{HW}((\hat{p}\check{p}+1)f) - \mathrm{HW}(e) \\ &\geq \mathrm{HW}(r) - \mathrm{HW}(\hat{p}\check{p}f) + \mathrm{HW}((\hat{p}\check{p}+1)f) - \mathrm{HW}(e)\ . \end{aligned}$$

Since $f \notin (\mathcal{F} \cup \{0,1\})$ Lemma 1 gives $\mathrm{HW}(\hat{p}\check{p}f) \leq \frac{d}{4}$, so $-\mathrm{HW}(\hat{p}\check{p}f) + \mathrm{HW}((\hat{p}\check{p}+1)f) \geq \frac{3}{4}d$. Therefore, we get $d(r+e,f) > \mathrm{HW}(r) + \frac{3}{4}d - \frac{1}{4}d = \mathrm{HW}(r) + \frac{1}{4}d$. On the other hand, the distance of $r+e$ to 0 is

$$\begin{aligned} d(r+e,0) &= \mathrm{HW}(r+e) \\ &\leq \mathrm{HW}(r) + \mathrm{HW}(e) \\ &< \mathrm{HW}(r) + \frac{1}{4}d\ . \end{aligned}$$

Therefore, if Algorithm 6 terminates in the for-loop of Line 2, the outcome of the ML decoder cannot be a word of $\bar{\mathcal{F}} \cup \{1\}$, which implies that $\mathfrak{D}_{\mathcal{RM}}(r) \in \mathcal{F}$. Now consider the case where Algorithm 6 terminates in the for-loop of Line 7. Note that, by definition of the sets $\hat{\mathcal{I}}$ and $\check{\mathcal{I}}$, we have $\mathrm{supp}(r) \subset \mathrm{supp}(\hat{p})$ and, since the for-loop of Line 2 is completed, it holds that $\mathrm{HW}(\hat{p}\check{p}r) = \frac{d}{2}$. To begin, observe that

$$\begin{aligned} d(r+e,\hat{p}) &= \mathrm{HW}(\hat{p}+r+e) \\ &\geq \mathrm{HW}(\hat{p}+r) + \mathrm{HW}(e) \\ &\overset{(\mathsf{a})}{=} d - \mathrm{HW}(r) + \mathrm{HW}(e) < \frac{5}{4}d - \mathrm{HW}(r)\ , \end{aligned} \tag{1}$$

where (a) holds because $\mathrm{supp}(r) \subset \mathrm{supp}(\hat{p})$ and $\mathrm{HW}(\hat{p}) = d$. It follows immediately from Lemma 1 that an $\mathcal{RM}(m)$ code can be partitioned by

$$\begin{aligned} \mathcal{RM}(m) = \{0\} \cup \{1\} \cup \{\hat{p}+1\} \cup \mathcal{F} \cup \{f \mid \mathrm{HW}(\hat{p}\check{p}f) = 0, \mathrm{HW}(\hat{p}(\check{p}+1)f) = \frac{d}{2}\} \\ \cup \{f \mid \mathrm{HW}(\hat{p}\check{p}f) = \frac{d}{4}, \mathrm{HW}(\hat{p}(\check{p}+1)f) = \frac{d}{4}\}\ . \end{aligned}$$

The statement holds if the distance to the words in all subsets except $\{0\}$ and $\mathcal{F}$ is larger than Eq. (1). We consider each subset separately:

- For $f = 1$ we have

$$\begin{aligned} d(r+e,f) &= \mathrm{HW}(f+r+e) \\ &\geq \mathrm{HW}(f) - \mathrm{HW}(r) - \mathrm{HW}(e) \\ &> 2d - \mathrm{HW}(r) - \frac{d}{4} \end{aligned}$$

$$> \frac{7}{4}d - \mathrm{HW}(r) > d(r+e,\hat{p}) \ .$$

– For $f = \hat{p} + 1$ we have

$$\begin{aligned} d(r+e,f) &= \mathrm{HW}(f+r+e) \\ &\geq \mathrm{HW}(f+r) - \mathrm{HW}(e) \\ &= \mathrm{HW}(\hat{p}(f+r)) + \mathrm{HW}((\hat{p}+1)(f+r)) - \mathrm{HW}(e) \\ &= \mathrm{HW}(\hat{p}r) + \mathrm{HW}((\hat{p}+1)f) - \mathrm{HW}(e) \\ &= \mathrm{HW}(r) + \mathrm{HW}(f) - \mathrm{HW}(e) \\ &= 2\mathrm{HW}(r) + d - \mathrm{HW}(r) - \mathrm{HW}(e) \\ &\overset{(\mathsf{a})}{\geq} 2d - \mathrm{HW}(r) - \mathrm{HW}(e) \\ &> \frac{7}{4}d - \mathrm{HW}(r) > d(r+e,\hat{p}) \ , \end{aligned}$$

where (a) holds because $\mathrm{HW}(r) \geq \mathrm{HW}(\hat{p}\check{p}r) = \frac{d}{2}$, as noted above.

– For any $f \in \mathcal{RM}(m)$ with $\mathrm{HW}(\hat{p}\check{p}f) = 0$ and $\mathrm{HW}(\hat{p}(\check{p}+1)f) \geq \frac{d}{4}$ we have

$$\begin{aligned} d(r+e,f) &= \mathrm{HW}(f+r+e) \\ &\geq \mathrm{HW}(f+r) - \mathrm{HW}(e) \\ &= \mathrm{HW}(\hat{p}\check{p}(f+r)) + \mathrm{HW}((\hat{p}\check{p}+1)(f+r)) - \mathrm{HW}(e) \\ &\geq \mathrm{HW}(\hat{p}\check{p}r) + \underbrace{\mathrm{HW}((\hat{p}\check{p}+1)f)}_{=\mathrm{HW}(f)=d} - \mathrm{HW}((\hat{p}\check{p}+1)r) - \mathrm{HW}(e) \\ &= d + \mathrm{HW}(\hat{p}\check{p}r) - \mathrm{HW}((\hat{p}\check{p}+1)r) - \mathrm{HW}(e) \\ &= d + \underbrace{2\mathrm{HW}(\hat{p}\check{p}r)}_{=d} - (\mathrm{HW}(\hat{p}\check{p}r) + \mathrm{HW}((\hat{p}\check{p}+1)r)) - \mathrm{HW}(e) \\ &= 2d - \mathrm{HW}(r) - \mathrm{HW}(e) \\ &> \frac{7}{4}d - \mathrm{HW}(r) > d(r+e,\hat{p}) \ . \end{aligned}$$

– For any $f \in \mathcal{RM}(m)$ with $\mathrm{HW}(\hat{p}\check{p}f) = \mathrm{HW}(\hat{p}(\check{p}+1)f) = \frac{d}{4}$ we have

$$\begin{aligned} d(r+e,f) &= \mathrm{HW}(f+r+e) \\ &\geq \mathrm{HW}(f+r) - \mathrm{HW}(e) \\ &= \mathrm{HW}(\hat{p}\check{p}(f+r)) + \mathrm{HW}(\hat{p}(\check{p}+1)(f+r)) + \mathrm{HW}((p_1+1)(f+r)) \\ &\quad - \mathrm{HW}(e) \\ &= \mathrm{HW}(\hat{p}\check{p}(f+r)) + \mathrm{HW}(\hat{p}(\check{p}+1)f) - \mathrm{HW}(\hat{p}(\check{p}+1)r) \\ &\quad + \underbrace{\mathrm{HW}((p_1+1)f)}_{=\frac{d}{2}} - \mathrm{HW}(e) \\ &= \frac{d}{4} + \frac{d}{4} - \mathrm{HW}(\hat{p}(\check{p}+1)r) + \frac{d}{2} - \mathrm{HW}(e) \end{aligned}$$

$$
\begin{aligned}
&= d + \mathrm{HW}(\hat{p}\check{p}r) - (\mathrm{HW}(\hat{p}\check{p}r) + \mathrm{HW}(\hat{p}(\check{p}+1)r)) - \mathrm{HW}(e) \\
&= \frac{3}{2}d - \mathrm{HW}(r) - \mathrm{HW}(e) \\
&> \frac{5}{4}d - \mathrm{HW}(r) > d(r+e,\hat{p}) \ .
\end{aligned}
$$

We conclude that for any $f \in \mathcal{RM} \setminus (\mathcal{F} \cup \{0\})$ a word of $\mathcal{F}$ (specifically $\hat{p}$) is closer[5] to $r+e$ than f, and it follows that $\mathfrak{D}_{\mathcal{RM}}(r+e) \in \mathcal{F}$. Since the distance to the word of $\mathcal{F}$ was truly smaller in each of the discussed cases, i.e., not a tie, the decision is not the result of a tie in the distance with some other word $\mathcal{RM}(m) \setminus \mathcal{F}$. □

Due to the specific structure of the words in the set $\mathcal{F}$, i.e., the possible outputs of an ML decoder for the considered input, we are now able to make a statement on the behavior of the oracle when a single bit of this input is flipped.

Lemma 3. *Denote by $\hat{p}, \check{p} \in \mathbb{F}_2[\boldsymbol{x}]$ two polynomials with $\deg(\hat{p}) = \deg(\check{p}) = 1$ and $\check{p} \notin \{\hat{p}, \hat{p}+1\}$. Then, for $r = \mathsf{FindWordAtBorder}(\mathfrak{D}_0^{\boldsymbol{e}}, \mathrm{supp}(\hat{p}), \mathrm{supp}(\check{p}))$ as in Algorithm 6 and any $\xi \in \mathrm{supp}(\hat{p}\check{p})$ it holds that*

$$
\mathfrak{D}_0^{\boldsymbol{e}}(r + u^{(\xi)}) = \begin{cases} \mathsf{True}, & \text{if } r_\xi + e_\xi = 1 \\ \mathsf{False}, & \text{else}, \end{cases}
$$

where $u^{(\xi)} \in \mathbb{F}_2^{2^m}$ denotes (polynomial corresponding to) the ξ-th unit vector.

Proof. Denote $\mathcal{F} = \{\hat{p}, \check{p}, \hat{p}+\check{p}+1\}$. By Lemma 2, we have $\mathfrak{D}_{\mathcal{RM}}(r+e) =: \tilde{p} \in \mathcal{F}$ for the word r returned at Step 4 of Algorithm 7. By definition of $\mathcal{F}$, this implies that $(\hat{\mathcal{I}} \cap \check{\mathcal{I}}) \subset \mathrm{supp}(\tilde{p})$, i.e., the positions $\hat{\mathcal{I}} \cap \check{\mathcal{I}}$ of $\tilde{p}$ are all one. Therefore, if a position in $\hat{\mathcal{I}} \cap \check{\mathcal{I}}$ of $r+e$ is changed from 0 to 1, the distance to $\tilde{p}$ *always* decreases by 1 and the ML decoder output does not change. On the other hand, if a position in $\hat{\mathcal{I}} \cap \check{\mathcal{I}}$ of $r+e$ is changed from 1 to 0, the distance to any polynomial of $\mathcal{F}$ *always* increases by 1, the distance to 0 decreases by 1, and the distance to any other word in $\mathcal{RM}(m) \setminus (\mathcal{F} \cup \{0\})$ decreases by at most 1. Hence, the ML decoding result changes from $\tilde{p}$ to 0 and the oracle returns True. □

Finally, we show that Algorithm 7 is always successful in recovering the correct vector e, given that some non-restrictive assumptions are fulfilled.

Theorem 1. *Let $\mathfrak{D}_0^{\boldsymbol{e}}$ be a oracle for the code $\mathcal{RM}^{\times s}(m) \subset \mathbb{F}_2^{s2^m}$ of minimum distance $d = s2^{m-1}$, where $\boldsymbol{e} \in \mathbb{F}_2^{s2^m}$ with $\mathrm{HW}(\boldsymbol{e}) < \frac{d}{4}$. Consider two polynomials $p_1, p_2 \in \mathbb{F}_2[\boldsymbol{x}]$ with $\deg(p_1) = \deg(p_2) = 1$ and $p_2 \notin \{p_1, p_1+1\}$. Then, the output of Algorithm 7 is $\mathsf{FindError}(\mathfrak{D}_0^{\boldsymbol{e}}, \mathrm{supp}^{\times s}(p_1), \mathrm{supp}^{\times s}(p_2)) = \boldsymbol{e}$.*

Proof. For sake of readability and ease of notation, we focus on the case of multiplicity $s = 1$ in this proof. It is easy to verify that all statements also hold

[5] Similarly to the previous case, this does not mean that the ML decoding result is necessarily $\hat{p}$, since the proof does not hold for $\check{p}$ and $\hat{p}+\check{p}+1$.

for $s > 1$ by essentially multiplying every weight/distance by s. Note that both Algorithms 6 and 7 are independent of s. Consider some choice of $\hat{\mathcal{I}}$ and $\check{\mathcal{I}}$ in Steps 2 and 3 of Algorithm 7. Note that there exist corresponding polynomials $\hat{p} \in \{p_1, p_1 + 1\}$ and $\check{p} \in \{p_2, p_2 + 1\}$ with $\mathrm{supp}(\hat{p}) = \hat{\mathcal{I}}$ and $\mathrm{supp}(\check{p}) = \check{\mathcal{I}}$ and we have $\deg(\hat{p}) = \deg(\check{p}) = 1$ and $\check{p} \notin \{\hat{p}, \hat{p} + 1\}$ for any such choice. Step 6 iterates over all positions of $\boldsymbol{r}$ in $\hat{\mathcal{I}} \cap \check{\mathcal{I}}$ and queries the oracle with this bit flipped. If this changes the oracle output to True, the corresponding bit is flipped in $\hat{\boldsymbol{e}}$, with the goal of obtaining $\hat{\boldsymbol{e}}_{\hat{\mathcal{I}} \cap \check{\mathcal{I}}} = \boldsymbol{e}_{\hat{\mathcal{I}} \cap \check{\mathcal{I}}}$ at the end of the loop. We consider the four different possible combinations of $\boldsymbol{e}_\xi$ and $\boldsymbol{r}_\xi$:

- $e_\xi = 0, r_\xi = 0$ or $e_\xi = 1, r_\xi = 1$: Flipping positions r_ξ corresponds to setting a 0 in $\boldsymbol{r} + \boldsymbol{e}$ to 1. By Lemma 3, this does not change the ML decoding result, i.e., the oracle still returns False. The bit $\hat{e}_\xi$ is not flipped, i.e., we have $\hat{e}_\xi = r_\xi$, and we correctly obtain $\hat{e}_\xi = e_\xi$.
- $e_\xi = 0, r_\xi = 1$ or $e_\xi = 1, r_\xi = 0$: Flipping positions r_ξ corresponds to setting a 1 in $\boldsymbol{r} + \boldsymbol{e}$ to 0. By Lemma 3, this does change the ML decoding result to all-zero, i.e., the oracle now returns True. The bit $\hat{e}_\xi$ is flipped, i.e., we have $\hat{e}_\xi = r_\xi + 1$, and we correctly obtain $\hat{e}_\xi = e_\xi$.

We conclude that $\tilde{\boldsymbol{e}}_{\hat{\mathcal{I}} \cap \check{\mathcal{I}}} = \hat{\boldsymbol{e}}_{\hat{\mathcal{I}} \cap \check{\mathcal{I}}} = \boldsymbol{e}_{\hat{\mathcal{I}} \cap \check{\mathcal{I}}}$. This holds for any choice of $\hat{\mathcal{I}}$ and $\check{\mathcal{I}}$. The lemma statement follows from observing that the corresponding sets $\hat{\mathcal{I}} \cap \check{\mathcal{I}}$ partition the set of all positions $[0, q^m - 1]$.

□

Required Oracle Calls Our strategy as described in Algorithm 7 requires at most $4 \cdot (\frac{2 \cdot n_2}{4} + \frac{n_2}{4})$ oracle calls dependent on the length of the RM code. Note that the algorithm has to be repeated for all n_1 blocks of $\boldsymbol{y}^{(0)}$ introducing an additional factor of n_1.

We observed that the attack strategy of exploiting shown by Guo et al. [3] can be adapted for the use with our oracle. In addition to some disadvantages of the exploited timing side-channel (see Section 4 for a detailed discussion) this approach shows a largely increased number of required oracle calls. In essence their attack works by randomly increasing the Hamming weight of an input to the RM decoder until they reach the decoding boundary. Then the oracle can be queried with the individual bits of the input flipped. From the now found error positions only those that are not self introduced in the first step are counted as a valid part of the support of $\boldsymbol{y}_i^{(0)}$. Therefore, the attack steps have to be repeated until each position is evaluated and optionally a certain threshold for each position is reached. This makes the attack non-deterministic, and therefore we simulated the required oracle calls for 400,000 attacks with $y_{w,max}$ in the range $1 \leq y_{w,max} \leq 10$ given a threshold of one (each position has to be evaluated once). The resulting mean of the required oracle calls in comparison with our strategy is shown in Table 3. To summarize our attack strategy in comparison to [3] requires by a factor of 11.73 (HQC-128) and 12.62 (HQC-192/256) less oracle queries. In addition, it is proven to be successful for $\mathrm{HW}(\boldsymbol{y}_i^{(0)}) < \frac{d_{\mathcal{RM}}}{4}$, where $d_{\mathcal{RM}} = s \cdot 2^{m-1}$.

	Timing Attack [3]	This work
HQC-128	13522*46	1152*46
HQC-192	24216*56	1920*56
HQC-256	24216*90	1920*90

Table 3: Comparison of required oracle queries for the different attack strategies. Note that the size of the RM code is the same for HQC-192 and HQC-256, and therefore we use the same simulation result for both parameter sets. Using the timing side-channel of [3] requires even more oracle calls as a majority decision or each bit has to be found in order to increase the reliability of the oracle.

3.4 Retrieval of y from Partial Information with Information Set Decoding

In the case that $\boldsymbol{y}^{(0)}$ is only partially retrieved we can still use this partial information to mount an attack through information set decoding. For a general approach on how to incorporate partial side-channel information into information-set decoding we refer the reader to [4]. There are two main reasons for the information to be limited. Either there is a limit on the amount of possible oracle calls due to the amount of decryptions that can be observed by the attacker or the side-channel used to create the oracle does not result in perfect oracle answers. In Appendix B we describe a modified variant of Stern's algorithm [14] that is able to incorporate correct information about the support of the individual $\boldsymbol{y}_i^{(0)}$ to lower the complexity of information set decoding. The resulting work factor $\mathcal{W}_{\text{ModSt}}$ given the knowledge of τ elements of the support of $\boldsymbol{y}^{(0)}$ is shown in Fig. 2. Note that our algorithm also uses the information whether a full block $\boldsymbol{y}_i^{(0)}$ has been retrieved, and we therefore assume the support of $\boldsymbol{y}^{(0)}$ is evenly distributed between the different blocks of $\boldsymbol{y}_i^{(0)}$ in Fig. 2.

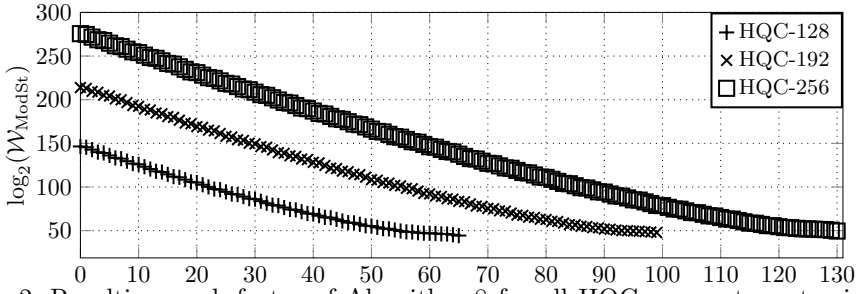

Fig. 2: Resulting work factor of Algorithm 8 for all HQC parameter sets given the knowledge of τ elements of the support of $\boldsymbol{y}^{(0)}$.

4 Side-channel Targets to Build the Required Oracle

There are several possible side-channels that can be used to construct our close-to-zero oracle as given in Definition 2. In this section we describe our results of directly attacking the implemented RS decoder of the HQC round 3 reference

implementation using our power side-channel described in [13]. In addition, we discuss the approaches in related work and show how or whether these side-channels can be adapted to build our oracle. An overview of the different side-channel targets of the HQC decapsulation is shown in Fig. 3. Note that we consider a discussion of the fault-attack of Xagawa et al. [17] out of scope for this work.

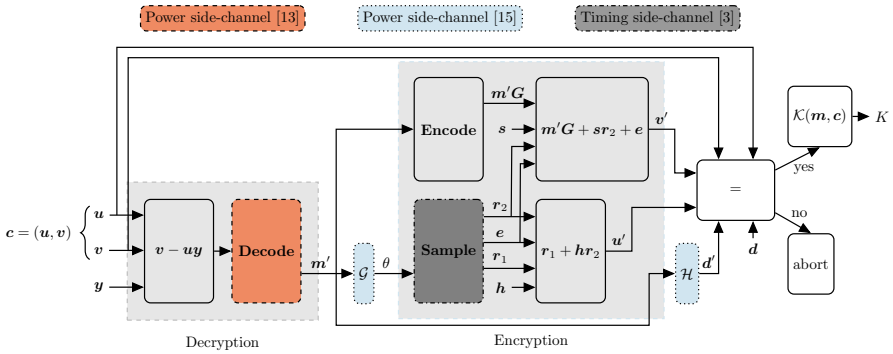

Fig. 3: Building blocks of the HQC decapsulation (c.f. Algorithm 5) with the side-channel attack targets used in related work.

4.1 Power Side-channel on the RS Decoder

It is possible to construct our required oracle from the decoding result of the RS decoder. First we have to recall that our oracle indicates whether the RM decoder is able to correctly decode to the all-zero codeword or the decoding fails, and any other codeword is returned. Transferring this behavior to be observable through the RS decoder requires to set all the remaining $\boldsymbol{y}_i^{(0)}$ that are not attacked to zero. Then if the RS decoder has to correct an error we know that the RM decoder was not able to return the correct all-zero codeword and the oracle result is True.

In order to observe that the RS decoder has to correct an error we use the template matching approach shown in [13]. Note that although this method is targeting a BCH decoder we can still use the approach in our setting. This is due to the fact that BCH codes are subcodes of RS codes and usually decoded using the same procedure as RS codes[6]. Therefore, steps during the decoding of these codes are essentially the same. The attack target for building the templates is the computation of the error-locator polynomial, as it showed the highest amount of exploitable leakage. As the input can be directly controlled by an attacker, templates for both classes can be constructed through the power side-channel. Then for each oracle call the constructed input is fed into the decapsulation and

[6] A (linear) subcode consists of a subset of codewords of the original code, usually in the form of a linear subspace. It is easy to see that any decoder for the original code can also be applied to the subcode, since it contains any codeword of the subcode. For more details on the relation between RS and BCH codes, the reader is referred to [7, Chapter 12].

the respective power trace is compared to the templates through the use of a sum-of-squared-difference metric. As a result the class with the smallest metric is chosen as the oracle result.

Attack Results We evaluated the oracle with our power side-channel setup consisting of a STM32F415RGT6 ARM Cortex-M4 microcontroller mounted on a Chipwhisperer CW308 UFO board running at 10 MHz. The power consumption is measured through the integrated shunt resistor with a PicoScope 6402D USB-oscilloscope running at a sampling rate of 156.25 MHz. As an attack target we use the latest version of the HQC-128 reference implementation. With our setup a total amount of 1000 template traces are used for the initialization of the oracle using a t-test threshold of 100 for point of interest detection. Using the initialized oracle we are able to correctly classify 100,000 attack traces. As this number is above the required number of oracle calls for a complete key recovery we consider an attack on the complete secret key successful.

4.2 Power Side-Channel on the used Hash Functions $\mathcal{G}, \mathcal{H}$

In [15] the authors show how to create a plaintext-checking oracle for HQC by observing a power side-channel of the used hash functions SHAKE256-512. With their oracle they are able to distinguish if a certain input results in a fixed message $\boldsymbol{m}'$ or if the result is different to this fixed message. As $\boldsymbol{m}'$ is directly used as an input to $\mathcal{G}$ and $\mathcal{H}$ the authors identify these hash functions as an attack target. In order to instantiate their oracle they use a machine learning classifier based on a convolutional neural network (CNN). They evaluate their CNN on the SHAKE software implementation of `pqm4` [6] with the same side-channel platform and microcontroller as we described in the previous section. Using 30,000 training traces they achieve an accuracy of 0.998 when classifying 10,000 test traces, which can be further increased through the combination of multiple classifications.

This oracle can not be directly used with our proposed attack strategy as the resulting message after decryption is always zero. It can nevertheless be adapted to work as our close-to-zero oracle (Definition 2) by setting the resulting input to **Decode** such that the input to the RS decoder is set to its decoding boundary. This can be done by setting $(\boldsymbol{d}_{\mathcal{RS}} - 1)/2$ blocks of $\boldsymbol{y}^{(0)}$, that are not currently attacked, to be decoded as a non-zero value and therefore acting as an error for the all-zero RS codeword. Then the RM decoding result of the attacked $\boldsymbol{y}_i^{(0)}$ determines if the resulting message is zero (True) or unequal to zero (False), which is observable through their oracle.

4.3 Timing Side-Channel of the used Sampler

Guo et al. [3] showed a timing side-channel in the implementation of the sampler of the HQC reference implementation that is used to generate the random fixed-weight vectors $\boldsymbol{e}$, $\boldsymbol{r}_1$, and $\boldsymbol{r}_2$. This is the case as the sampler implements rejection sampling, which requires a varying amount of calls to the PRNG in order to generate potential required additional randomness. As the seed θ for the

PRNG is derived from the message $\boldsymbol{m}$, the amount of additional required PRNG calls is dependent on $\boldsymbol{m}$ and therefore also the execution time of the decapsulation. This timing side-channel allows to build a plaintext-checking oracle for which the authors show an attack strategy. In order for two different messages to be distinguishable through their oracle the initial message is chosen such that it requires at least three additional calls to the PRNG which has a low probability of 0.58 for all possible messages. Due to the inherent uncertainty of timing side-channels and this probability still leaving room for ambiguity the authors introduce a majority threshold for the classification of each bit. Their empirical results show a classification success rate of 87% with a majority threshold of five.

Their oracle can not be used with our attack strategy as its resulting message $\boldsymbol{m}$ is always zero. This message unfortunately does not require multiple calls to the PRNG, and therefore it is not easily distinguishable through this timing side-channel. In contrast, our developed attack strategy allows the usage of both described power side-channels, which show a better classification result and eliminate the inherent uncertainty of timing side-channels removing the need for a majority decision.

5 Conclusion

In this paper we showed a novel proven side-channel attack strategy that again allows for a successful power side-channel attack against the updated round three version of the HQC cryptosystem. Published attacks against the former HQC version are not valid anymore, as the authors updated their used error correcting codes for their third round submission. We identified that the published power side-channel attack on the updated HQC version by Uneo et al. [15] is not valid in practice, as the authors miss a crucial property of the implemented Reed-Muller decoder that renders their attack unsuccessful. In contrast to the attack strategy by Guo et al. [3], that exploit a timing side-channel in the implemented polynomial sampler, our attack shows a by a factor of 12 reduced amount of required side-channel oracle calls. Our attack strategy allows the use of two power side-channel targets, namely the Reed-Solomon decoder as shown in [13] and the used SHAKE256 hash function as described in [15], to build the required oracle for our attack. We show practical attack results for the latest Reed-Solomon decoder of the latest HQC-128 implementation on an ARM Cortex-M4 microcontroller. Finally, we provided an estimation of the remaining attack complexity for partial attack results with the use of information set decoding.

Acknowledgment

This work was supported by the German Research Foundation (Deutsche Forschungsgemeinschaft, DFG) under Grant No. SE2989/1-1 as well as WA3907/4-1 and the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 801434). We would like to thank Christoph Frisch for his helpful feedback.

References

1. Albrecht, M.R., et al.: NIST post-quantum cryptography standardization round 3 submission: Classic McEliece, https://classic.mceliece.org
2. Bâetu, C., Durak, F.B., Huguenin-Dumittan, L., Talayhan, A., Vaudenay, S.: Misuse attacks on post-quantum cryptosystems. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 747–776. Springer (2019)
3. 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, Report 2021/1485 (2021), https://ia.cr/2021/1485
4. Horlemann, A.L., Puchinger, S., Renner, J., Schamberger, T., Wachter-Zeh, A.: Information-set decoding with hints. In: Code-Based Cryptography, pp. 60–83. Springer International Publishing (2022)
5. Huguenin-Dumittan, L., Vaudenay, S.: Classical misuse attacks on NIST round 2 PQC: The power of rank-based schemes
6. Kannwischer, M.J., Rijneveld, J., Schwabe, P., Stoffelen, K.: PQM4: Post-quantum crypto library for the ARM Cortex-M4, https://github.com/mupq/pqm4
7. MacWilliams, F.J., Sloane, N.J.A.: The theory of error correcting codes, vol. 16. Elsevier (1977)
8. Melchor, C.A., et al.: NIST post-quantum cryptography standardization round 2 submission: Hamming Quasi-Cyclic (HQC), http://pqc-hqc.org/
9. Melchor, C.A., et al.: Nist post-quantum cryptography standardization round 3 submission: Hamming Quasi-Cyclic (HQC), http://pqc-hqc.org/
10. Moody, D., et al.: Status report on the second round of the NIST post-quantum cryptography standardization process. Tech. rep. (jul 2020). https://doi.org/10.6028/nist.ir.8309
11. Paiva, T.B., Terada, R.: A timing attack on the HQC encryption scheme. In: Paterson, K.G., Stebila, D. (eds.) Selected Areas in Cryptography – SAC 2019. pp. 551–573. Springer International Publishing, Cham (2020)
12. Renner, J.: Post-Quantum Cryptography in the Hamming Metric, the Rank Metric, and the Sum-Rank Metric. Dissertation (Ph.D. thesis), Technische Universität München (2022)
13. 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. pp. 119–134. Springer (2020)
14. Stern, J.: A method for finding codewords of small weight. In: Cohen, G., Wolfmann, J. (eds.) Coding Theory and Applications. pp. 106–113. Springer Berlin Heidelberg, Berlin, Heidelberg (1989)
15. Ueno, R., Xagawa, K., Tanaka, Y., Ito, A., Takahashi, J., Homma, N.: Curse of re-encryption: A generic power/em analysis on post-quantum KEMs. IACR Transactions on Cryptographic Hardware and Embedded Systems **2022**(1), 296–322 (Nov 2021), https://tches.iacr.org/index.php/TCHES/article/view/9298
16. Wafo-Tapa, G., Bettaieb, S., Bidoux, L., Gaborit, P., Marcatel, E.: A practicable timing attack against HQC and its countermeasure. Cryptology ePrint Archive, Report 2019/909 (2019), https://eprint.iacr.org/2019/909
17. Xagawa, K., Ito, A., Ueno, R., Takahashi, J., Homma, N.: Fault-injection attacks against NIST's post-quantum cryptography round 3 KEM candidates. Cryptology ePrint Archive, Report 2021/840 (2021), https://ia.cr/2021/840

A Counterexample to the Attack Strategy in [15,17]

The work [2, Figure 7] presents a learning algorithm which allows for determining an additive error given access to a decode-to-zero oracle for a bounded-distance decoder. In [2, Theorem 11] it is shown that this algorithm succeeds with probability 1 in the considered setting. Given the vectors $\boldsymbol{r}$ and $\boldsymbol{e}$, the oracle is defined as

$$\mathsf{BOO}(\boldsymbol{r}) = \begin{cases} \mathsf{True}, & \text{if } \mathrm{HW}(\boldsymbol{r}+\boldsymbol{e}) \leq \tau, \\ \mathsf{False}, & \text{else.} \end{cases}$$

In other words, the oracle provides the information whether the sum of the input $\boldsymbol{r}$ and the error $\boldsymbol{e}$ would be corrected to zero by a bounded-distance decoder of radius τ. Note that $\mathsf{BOO}(\boldsymbol{r})$ is similar to the oracle $\mathfrak{D}_0^{\boldsymbol{e}}(\boldsymbol{r})$ in our work, as given in Definition 2, except that it assumes a bounded-distance decoder, i.e., a fixed decoding threshold, instead of an ML decoder. In [17, Section C.7][7] it is claimed that this algorithm can be applied to the round three version of HQC system, which employs an ML decoder. However, the fixed decoding threshold of the bounded-distance decoder is essential to the algorithm of [2, Figure 7] and in this section we show that replacing the $\mathsf{BOO}(\boldsymbol{r})$ oracle with the $\mathfrak{D}_0^{\boldsymbol{e}}(\boldsymbol{r})$ oracle, i.e., considering an ML decoder instead of a bounded-distance decoder, causes this algorithm to return incorrect error vectors. Note that this choice of decoder is inherent to the system and cannot be influenced by the attacker, so the oracle $\mathfrak{D}_0^{\boldsymbol{e}}(\boldsymbol{r})$ is the appropriate oracle to use for this system. In addition to this counterexample we implemented the attack strategy and performed a simulated attack by directly accessing the RM decoder results of the HQC reference implementation. We were not able to correctly retrieve the support $\boldsymbol{y}_i^{(0)}$ with our simulations.

The algorithm of [2, Figure 7] is based on constructing a vector, such that the sum of this vector and the error is at the decoding threshold. For a bounded-distance decoder, the result of the BOO oracle, when queried with a single bit of this input flipped, then determines the corresponding position of the error vector. However, this only applies if the result of input x plus error e is at the decoding threshold *for every position*. We give an example of a vector for which this is only the case for a subset of positions, which leads to an incorrect output, even in the case of a single error. We follow the steps of the algorithm of [2, Figure 7] and fix the error vector to be the first unit vector $\boldsymbol{e} = \boldsymbol{e}^{(1)}$.

$$\begin{aligned}
&\text{Initialize} \\
&\quad x \quad (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0) \\
&\quad y \quad (1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1) \\
&\text{First while loop iteration} \\
&\quad u \quad (1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0) \\
&\quad v \quad (0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1) \\
&\qquad \mathfrak{D}_0^{\boldsymbol{e}}(x+u) = \mathsf{False} \\
&y \leftarrow u \quad (1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0)
\end{aligned}$$

[7] The authors of [15] directly cite [17] for their attack description.

Second while loop iteration

$$\begin{aligned} u &\ (0,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0) \\ v &\ (1,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0) \\ &\ \mathfrak{D}_0^{\boldsymbol{e}}(x+u) = \mathsf{False} \\ y \leftarrow u &\ (0,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0) \end{aligned}$$

Third while loop iteration

$$\begin{aligned} u &\ (0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0) \\ v &\ (0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0) \\ &\ \mathfrak{D}_0^{\boldsymbol{e}}(x+u) = \mathsf{True} \\ x \leftarrow x+u &\ (0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0) \\ y \leftarrow v &\ (0,0,0,1,1,0,0,0,0,0,0,0,0,0,0,0) \end{aligned}$$

Fourth while loop iteration

$$\begin{aligned} u &\ (0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0) \\ v &\ (0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0) \\ &\ \mathfrak{D}_0^{\boldsymbol{e}}(x+u) = \mathsf{True} \\ x \leftarrow x+u &\ (0,1,1,0,1,0,0,0,0,0,0,0,0,0,0,0) \\ y \leftarrow v &\ (0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0) \\ ||y|| = 1 &\ \text{Terminate while loop.} \end{aligned}$$

As claimed, we now have a word $x + \boldsymbol{e}$ that is at the threshold of decoding, i.e., if a position inside its support is flipped, we decode to zero, i.e., $\mathfrak{D}_0^{\boldsymbol{e}}(x) = \mathsf{True}$. The last for-loop of the algorithm iterates over all positions, checking if flipping each bit alters the decoding result. Initialize z to

$$z \leftarrow x \ \ (0,1,1,0,1,0,0,0,0,0,0,0,0,0,0,0) \ .$$

So, e.g., for the error position we have

$$\begin{aligned} x + e^{(1)} &\ \ (1,1,1,0,1,0,0,0,0,0,0,0,0,0,0,0) \\ x + e^{(1)} + \boldsymbol{e} &\ \ (0,1,1,0,1,0,0,0,0,0,0,0,0,0,0,0) \end{aligned}$$

and therefore $\mathfrak{D}_0^{\boldsymbol{e}}(x + e^{(1)}) = \mathsf{True}$. Hence, the first bit in the vector z is flipped to obtain

$$z \leftarrow z + \boldsymbol{e}^{(1)} \ \ (1,1,1,0,1,0,0,0,0,0,0,0,0,0,0,0) \ .$$

Similarly, for any other position in the support, such as, e.g., $i = 2$, we have

$$\begin{aligned} x + e^{(2)} &\ \ (1,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0) \\ x + e^{(2)} + \boldsymbol{e} &\ \ (0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0) \end{aligned}$$

and therefore $\mathfrak{D}_0^{\boldsymbol{e}}(x + e^{(2)}) = \mathsf{True}$, so the second bit of z is flipped to obtain

$$z \leftarrow z + \boldsymbol{e}^{(2)} \ \ (0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0) \ .$$

After the first 8 positions we have

$$z \leftarrow z + \boldsymbol{e}^{(1)} \ \ (1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0) \ .$$

However, take for example $i = 9$. Then,

$$\begin{array}{rl} x + e^{(9)} & (0,1,1,0,1,0,0,0,1,0,0,0,0,0,0,0) \\ x + e^{(9)} + \boldsymbol{e} & (1,1,1,0,1,0,0,0,1,0,0,0,0,0,0,0) \end{array} .$$

At this point the difference between a bounded-distance decoder and an ML decoder affects the decoding decision. While this word does have weight more than $d/2$, it is easy to check that an ML decoder still decides for the all-zero word, so $\mathfrak{D}_0^{\boldsymbol{e}}(x + e^{(9)}) = \mathsf{True}$ and we get

$$z \leftarrow z + \boldsymbol{e}^{(1)} \ \ (1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0) \ .$$

This holds for all positions in the second part of the word, so at the end of the algorithm the "approximated error vector" is given by

$$z \leftarrow z + \boldsymbol{e}^{(1)} \ \ (1,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1) \ .$$

Hence, even in this simple case of a single error, the strategy does not return the correct error vector.

B Modified Variant of Stern's Algorithm

Let $\mathcal{J} \coloneqq \{j_0, \ldots, j_{\tau-1}\} \subseteq \mathrm{supp}(\boldsymbol{y}) \subseteq [0\,{:}\,n-1]$ be the subset of the support of $\boldsymbol{y}$ that we retrieved, and let $\mathcal{L} \coloneqq \{l_0, \ldots, l_{\iota-1}\} \subseteq [0\,{:}\,n-1] \setminus \mathrm{supp}(\boldsymbol{y})$ denote the indices of the zero entries in $\boldsymbol{y}$ that we determined. Then, we us obtain the secret vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ using the modified variant of Stern's algorithm [14] that is shown in Algorithm 8.

Theorem 2. *Let n and w be parameters chosen according to Table 1. Let $(\boldsymbol{x}, \boldsymbol{y})$ and $(\boldsymbol{h}, \boldsymbol{s})$ be a private and public key pair generated by Algorithm 1 for the chosen parameters. Let $\mathcal{J} \coloneqq \{j_0, \ldots, j_{\tau-1}\}$ be a subset of the support of $\boldsymbol{y}$, let $\mathcal{L} = \{l_0, \ldots, l_{\iota-1}\}$ denote a set of indices of zero entries in $\boldsymbol{y}$, and let $n' = n - \iota$ and $w' = w - \tau$. Furthermore, let k_1, p_{St}, $\nu_{\mathrm{St},1}$, and $\nu_{\mathrm{St},2}$ be non-negative integers such that $k_1 \leq n'$, $p_{\mathrm{St}} \leq w + w'$, $\nu_{\mathrm{St},1} \leq n - k_1$, and $\nu_{\mathrm{St},2} \leq n - n' + k_1$.*

Then, given $\boldsymbol{H} = [\boldsymbol{1}, \mathrm{rot}(\boldsymbol{h})] \in \mathbb{F}_2^{n \times 2n}$, w, $\boldsymbol{s} \in \mathbb{F}_2^n$, k_1, p_{St}, $\nu_{\mathrm{St},1}$, $\nu_{\mathrm{St},2}$, $\mathcal{J}$, and $\mathcal{L}$, Algorithm 8 outputs the vector $[\boldsymbol{x}, \boldsymbol{y}]$ with, on average, approximately

$$\mathcal{W}_{\mathrm{ModSt}} \coloneqq \frac{\mathcal{W}_{\mathrm{St,Iter}}}{P_{\mathrm{St}}}$$

operations in $\mathbb{F}_2$*, where*

$$\mathcal{W}_{\mathrm{St,Iter}} \coloneqq (n+n')^3 + (\nu_{\mathrm{St},1}+\nu_{\mathrm{St},2})\left(\sum_{i=1}^{p_{\mathrm{St}}}\binom{M_1}{i}+\sum_{i=1}^{p_{\mathrm{St}}}\binom{M_2}{i}-n'+\binom{M_2}{p_{\mathrm{St}}}\right) + 2^{1-\nu_{\mathrm{St},1}-\nu_{\mathrm{St},2}}\binom{M_1}{p_{\mathrm{St}}}\binom{M_2}{p_{\mathrm{St}}}(w+w'-2p_{\mathrm{St}}+1)(2p_{\mathrm{St}}+1),$$

the quantities $M_1 = \lfloor k_1/2 \rfloor + \lfloor (n'-k_1)/2 \rfloor$ *and* $M_2 = \lceil k_1/2 \rceil + \lceil (n'-k_1)/2 \rceil$*, and*

$$P_{\mathrm{St}} \coloneqq \sum_{\substack{\boldsymbol{a}\in\mathbb{N}_0^2 \\ a_1\le w \\ a_2\le w' \\ a_1+a_2=p_{\mathrm{St}}}} \sum_{\substack{\boldsymbol{b}\in\mathbb{N}_0^2 \\ b_1\le w-a_1 \\ b_2\le w'-a_2 \\ b_1+b_2=p_{\mathrm{St}}}} \frac{\binom{\lfloor k_1/2\rfloor}{a_1}\binom{\lceil k_1/2\rceil}{b_1}\binom{n-k_1-\nu_{\mathrm{St},1}}{w-a_1-b_1}}{\binom{n}{w}} \frac{\binom{\lfloor (n'-k_1)/2\rfloor}{a_2}\binom{\lceil (n'-k_1)/2\rceil}{b_2}\binom{k_1-\nu_{\mathrm{St},2}}{w'-a_2-b_2}}{\binom{n'}{w'}}.$$

Proof. In Line 1, the algorithm uses $\mathcal{J}$ to transform the syndrome decoding instance $(\boldsymbol{H}, \boldsymbol{s}, [\boldsymbol{x}, \boldsymbol{y}])$ of length $2n$, dimension n and error weight $2w$ into the syndrome decoding instance $(\boldsymbol{H}, \tilde{\boldsymbol{s}}, [\boldsymbol{x}, \tilde{\boldsymbol{y}}])$ of length $2n$, dimension n and error weight $w+w'$, where $\mathrm{HW}(\tilde{\boldsymbol{y}}) = w'$ and $\mathrm{supp}(\tilde{\boldsymbol{y}}) \cup \mathcal{J} = \mathrm{supp}(\boldsymbol{y})$. The remaining steps are equal to the modification of Stern's algorithm presented in [4] and [12, Alg. 17] except that the set $\mathcal{X}_2$ always contains $\mathcal{L}_1 = \{l_0, \ldots, l_{\lceil \iota/2 \rceil - 1}\}$ and the set $\mathcal{Y}_2$ always contains $\mathcal{L}_2 = \{l_{\lceil \iota/2 \rceil}, \ldots, l_{\iota-1}\}$. By this choice of $\mathcal{X}_2$ and $\mathcal{Y}_2$, the syndrome decoding instance $(\boldsymbol{H}, \tilde{\boldsymbol{s}}, [\boldsymbol{x}, \tilde{\boldsymbol{y}}])$ is transformed into the $(\bar{\boldsymbol{H}}, \bar{\boldsymbol{s}}, [\boldsymbol{x}, \bar{\boldsymbol{y}}])$ instance of length $n+n'$, dimension n' and error weight $w+w'$, where $\bar{\boldsymbol{H}} \in \mathbb{F}_2^{n' \times (n+n')}$, $\bar{\boldsymbol{s}} \in \mathbb{F}_2^{n+n'}$, and $\bar{\boldsymbol{y}} \in \mathbb{F}_2^{n'}$.

Such that an iteration of Stern's algorithm can solve the $(\bar{\boldsymbol{H}}, \bar{\boldsymbol{s}}, [\boldsymbol{x}, \bar{\boldsymbol{y}}])$ syndrome decoding instance, there must be exactly p_{St} error positions in both $\mathcal{X}_1 \cup \mathcal{X}_2$ and $\mathcal{Y}_1 \cup \mathcal{Y}_2$ and no error positions in $\mathcal{Z}_1$ and $\mathcal{Z}_2$. The probability that this event occurs is equal to P_{St}, cf. [4] and [12, Thm. 4.9]. This implies that, on average, Lines 5 to 12 need to be executed $1/P_{\mathrm{St}}$, where each iteration has a complexity of $\mathcal{W}_{\mathrm{St,Iter}}$. □

Algorithm 8: Modified Stern Algorithm

Input : Parity-check matrix $\boldsymbol{H} \in \mathbb{F}_2^{n \times 2n}$
Non-negative integer w
Syndrome vector $\boldsymbol{s} \in \mathbb{F}_2^n$
Non-negative integers k_1, p_{St}, $\nu_{\mathrm{St},1}$, $\nu_{\mathrm{St},2}$
Subset of the support $\mathcal{J} \coloneqq \{j_0, \ldots, j_{\tau-1}\} \subseteq \operatorname{supp}(\boldsymbol{y})$
Subset of zero entries $\mathcal{L} = \{l_0, \ldots, l_{\iota-1}\} \subseteq [0\!:\!n-1] \setminus \operatorname{supp}(\boldsymbol{y})$

Output: Vector $[\boldsymbol{x}, \boldsymbol{y}] \in \mathbb{F}_2^{2n}$

$\tilde{\boldsymbol{s}} \leftarrow \boldsymbol{s} + \boldsymbol{h}_{n+j_0}^\top + \ldots + \boldsymbol{h}_{n+j_{\tau-1}}^\top \in \mathbb{F}_2^n$, where $\boldsymbol{h}_\ell$ is the ℓ-th column of $\boldsymbol{H}$
$\mathcal{L}_1 \leftarrow \{l_0, \ldots, l_{\lceil \iota/2 \rceil - 1}\}$
$\mathcal{L}_2 \leftarrow \{l_{\lceil \iota/2 \rceil}, \ldots, l_{\iota-1}\}$
$\boldsymbol{e}' \leftarrow \boldsymbol{0} \in \mathbb{F}_2^{2n}$
while $\mathrm{HW}(\boldsymbol{e}') > 2w - \tau \vee \tilde{\boldsymbol{s}} \neq \boldsymbol{e}'\boldsymbol{H}^\top$ **do**
  $\mathcal{X}_1 \xleftarrow{\$} \{\mathcal{S} \subseteq [0\!:\!n-1] : |\mathcal{S}| = \lfloor k_1/2 \rfloor\}$
  $\mathcal{Y}_1 \xleftarrow{\$} \{\mathcal{S} \subseteq [0\!:\!n-1] \setminus \mathcal{X}_1 : |\mathcal{S}| = \lceil k_1/2 \rceil\}$
  $\mathcal{Z}_1 \xleftarrow{\$} \{\mathcal{S} \subseteq [0\!:\!n-1] \setminus (\mathcal{X}_1 \cup \mathcal{Y}_1) : |\mathcal{S}| = \nu_{\mathrm{St},1}\}$
  $\mathcal{X}_2 \xleftarrow{\$} \{\mathcal{S} \cup \mathcal{L}_1 \subseteq [n\!:\!2n-1] \setminus \mathcal{L}_2 : |\mathcal{S} \cup \mathcal{L}_1| = \lfloor (n-k_1)/2 \rfloor\}$
  $\mathcal{Y}_2 \xleftarrow{\$} \{\mathcal{S} \cup \mathcal{L}_2 \subseteq [n\!:\!2n-1] \setminus \mathcal{X}_2 : |\mathcal{S} \cup \mathcal{L}_2| = \lceil (n-k_1)/2 \rceil\}$
  $\mathcal{Z}_2 \xleftarrow{\$} \{\mathcal{S} \subseteq [n\!:\!2n-1] \setminus (\mathcal{X}_2 \cup \mathcal{Y}_2) : |\mathcal{S}| = \nu_{\mathrm{St},2}\}$
  $\boldsymbol{e}' \leftarrow$ Iteration of original Stern algorithm w.r.t. the syndrome $\tilde{\boldsymbol{s}}$, the sets $\mathcal{X}_1 \cup \mathcal{X}_2$, $\mathcal{Y}_1 \cup \mathcal{Y}_2$, $\mathcal{Z}_1 \cup \mathcal{Z}_2$ and the parameters p_{St} and $\nu_{\mathrm{St}} = \nu_{\mathrm{St},1} + \nu_{\mathrm{St},2}$
return $\boldsymbol{e} \in \mathbb{F}_2^{2n}$ *with support* $\operatorname{supp} \boldsymbol{e}' \cup \mathcal{J}$