Single trace HQC shared key recovery with SASCA

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

[1] XLIM, University of Limoges, Limoges
[2] Univ. Grenoble Alpes, CEA, Leti, MINATEC Campus, F-38054 Grenoble, France

Abstract. This paper presents practicable single trace attacks against the Hamming Quasi-Cyclic (HQC) Key Encapsulation Mechanism. These attacks are the first Soft Analytical Side-Channel Attacks (SASCA) against code-based cryptography. We mount SASCA based on Belief Propagation (BP) on several steps of HQC's decapsulation process. Firstly, we target the Reed-Solomon (RS) decoder involved in the HQC publicly known code. We perform simulated attacks under Hamming weight leakage model, and reach excellent accuracies (superior to 0.9) up to a high noise level ($\sigma = 3$), thanks to a re-decoding strategy. In a real case attack scenario, on a STM32F407, this attack leads to a perfect success rate. Secondly, we conduct an analogous attack against the RS encoder used during the re-encryption step required by the Fujisaki-Okamoto-like transform. Both in simulation and practical instances, results are satisfactory and this attack represents a threat to the security of HQC. Finally, we analyze the strength of countermeasures based on masking and shuffling strategies. In line with previous SASCA literature targeting Kyber, we show that masking HQC is a limited countermeasure against BP attacks, as well as shuffling countermeasures adapted from Kyber. We evaluate the "full shuffling" strategy which thwarts our attack by introducing sufficient combinatorial complexity. Eventually, we highlight the difficulty of protecting the current RS encoder with a shuffling strategy. A possible countermeasure would be to consider another encoding algorithm for the scheme to support a full shuffling. Since the encoding subroutine is only a small part of the implementation, it would come at a small cost.

Keywords: Soft Analytical Side-Channel Attack (SASCA) · Belief Propagation (BP) · Hamming Quasi-Cyclic (HQC) · Post-Quantum Cryptography (PQC) · Single Trace · Shared key recovery · Reed-Solomon (RS) codes

Introduction

Hamming Quasi-Cyclic (HQC) [AMAB+17] is a code-based Key Encapsulation Mechanism (KEM) involved in the American National Institute of Standards and Technology (NIST) process for Post-Quantum Cryptography (PQC) standardization [CCJ+16]. After three preliminary rounds and the standardization of lattice-based cryptography, HQC, along with BIKE [ABB+17] and ClassicMcEliece [BCL+], is now a candidate of the fourth and last round [AAC+22].

During this contest, the security of involved cryptosystems has been extensively studied by the community. HQC has been the target of several Side-Channel Attacks (SCA) since 2019. The former version of HQC, based on BCH codes, was attacked by two resembling timing attacks [PT19, WTBB+20] in 2019 and by a chosen ciphertext attack [SRSWZ20] by Schamberger et al. in 2020. The latter attack is based on a decoding oracle that can

corresponding authors: {guillaume.goy,julien.maillard}@cea.fr

distinguish whenever the BCH decoder corrects an error. Thanks to a chosen ciphertext strategy along with a resolution based on linear algebra, they successfully recover the whole secret key. In 2022, authors adapted their approach to build an attack [SHR+22] against the new version of HQC based on concatenated Reed-Muller (RM) and Reed-Solomon (RS) codes, allowing successful recovery of the secret key with 50000 power traces. Meanwhile, another key recovery side-channel attack with chosen ciphertext strategy [GLG22a] was exhibited against HQC-RMRS. Authors targeted the Fast Hadamard Transformed (FHT), involved in the RM decoder, to perform an attack with less than 20000 electromagnetic measurements.

Eventually, Goy et al. exposed the first single trace attack targeting the HQC shared key [GLG22b]. They used the structure of the concatenated RMRS decoder and the Decryption Failure Rate (DFR) [AMAB+17] analysis to observe that, in practice, the RS decoder manipulates mostly error-free codewords. The idea behind the attack is interesting, but authors were unable to recover the shared key from the noisy side-channel information without computing at least 2^{96} algebraic operations. This paper shows a vulnerability in the implementation of HQC-RMRS, but does not propose a practical attack.

To be complete, HQC can also be the target of generic attacks [RRCB20, UXT+22] targeting the Fujisaki-Okamoto (FO) transform construction, cache attacks [HSC+23] and timing attacks exploiting the randomness generator, namely the rejection sampling [GHJ+22]. These attacks will not be detailed in this paper.

Soft Analytical Side-Channel Attacks (SASCA) are powerful methods to perform SCA. SASCA algorithms are mostly based on Belief Propagation (BP) theory, which details can be found in [Mac03], chapter 26. BP was first used as SCA against cryptography by Veyrat-Charvillon et al. [VCGS14] in 2014, targeting the AES Furious implementation. Authors described a practical attack and emphasize on the efficiency of SASCA compared with the best state-of-the-art attacks at the time. SASCA was also used against the standardized cryptographic hash function Keccak: in 2020, Kannwischer et al. [KPP20] described a single trace attack on SHA-3. Authors mentioned a boolean masking countermeasure to thwart the attack, however, as specify in [GS18], masking countermeasures could enable new attacks.

Finally, SASCA was also applied on PQC, namely the standardized lattice-based KEM Kyber [BDK+18], renamed Module-Lattices KEM (ML-KEM) by the FIPS 203 [oSU23], was the target of four attacks [PPM17, PP19, HHP+21, HSST23] between 2017 and 2023. Primas et al. [PPM17] introduced the first BP based attack against Kyber. They showed that SASCA could be mounted against lattice-based cryptography, targeting the Number Theoretic Transform (NTT), an optimization strategy for lattice-based cryptography. Furthermore, they target a masked implementation of the NTT, leading at always recovering the secret key in real case attack scenario. Authors performed the attack in simulations under a Hamming weight leakage model, and obtained a satisfactory success rate (superior to 0.9) up to a $\sigma = 0.4$ noise level. Their evaluation on a real device required to build around one million templates. Later, Pessl and Primas [PP19] improved the attack by only crafting 213 Hamming weight templates. They also use node-merging (to limit the number of cycles), damping and graph scheduling techniques. Simulations showed a good success rate up to $\sigma = 1.5$. In 2021, Hamburg et al. [HHP+21] combined SASCA with a Chosen Ciphertext Attack (CCA) strategy, recovering the long-term secret key up to $\sigma = 2$ with a success rate superior to 0.9. Ravi et al. [RPBC20] introduced fine and coarse shuffling countermeasures to thwart BP attacks. In 2023, Hemerlink et al. [HSST23] analyzed the strength of these shuffling countermeasures and assessed their resistance against Hamburg et al. attack. So far, these shuffling countermeasures were not threatened by any attacks, but authors emphasize that this situation could lead to a "false security perception", and encourage precaution.

Our contributions In this work, we introduce the first practical single trace belief propagation attack against a PQC code-based cryptosystem, HQC, that can be executed within a few minutes. Specifically we recover the shared key manipulated by the Reed-Solomon code involved in HQC-RMRS scheme. All presented attacks exploit either one or two templates targeting the Galois field multiplication. We show that the reference implementation of HQC can be targeted by single trace attacks, and still threatened when protected with some countermeasures. Our attacks are performed both in simulations and in a real attack scenario on a STM32F407.

- We first exploit the point of vulnerability identified by Goy et al. [GLG22b] and transform it into a practical single trace side-channel attack aiming at shared key recovery. While this attack is based on the establishment of prior templates, the requirement for BP strategies is highly dependent on implementation choices. We describe how to build the factor graph for the RS decoder algorithm, which manipulates the error-free codeword containing information about the shared key.

- We show that codeword masking [MSS13], a masking strategy applicable to HQC, does not provide satisfactory security against our attack. Even if simple masking countermeasures of Kyber's NTT have been shown vulnerable to SASCA attacks, this consideration cannot be applied as-is for HQC. Indeed, codeword masking of the RS decoder is performed with a RS encoder for performance purposes. Hence, we provide with a study against the RS encoder and show that no reasonable masking countermeasure can thwart our attack.

- We also study the strength of known shuffling strategies (fine and coarse) [RPBC20], along with HQC specific strategy (window shuffling) against our attack. We show that none of these strategies constitute a sustainable countermeasure against our attack in a real case attack scenario. From an idea of [ATT+18], we derive the "full shuffling" strategy. This allows adding a high combinatorial complexity, making the attack impractical.

- Eventually, we observe that the re-encryption from the FO-like transform implies an additional encoder call during the decapsulation process. We combine the encoder and decoder leakages to perform a decapsulation attack. This new attack strategy requires to protect both decoder and encoder to thwart the threat. We show that changing the RS encoder strategy allows using the full shuffling and protect against our attack.

Outline Section 1 recalls HQC construction, presents the targeted algorithms as well as the SASCA approach. Section 2 introduces the attacker model. Section 3 presents the single template attack, exploiting only the template leakages. Section 4 introduces the graph construction and SASCA attack against the RS decoder and presents our simulation attack results. Section 5 targets the codeword masking countermeasure for RS decoder, where we redo the same work as the previous section against the encoder. Section 6 presents practical attacks against the weak shuffling countermeasures (fine, coarse and window) along with evaluating the full shuffling. Section 7 introduces the decapsultation attack, combining leakages from decryption and re-encryption taking advantage of the FO-like structure. In Section 8, we illustrate practical results obtained in a real case scenario for each of our attacks. Eventually, we draw conclusions and perspectives in Section 9.

1 Background

1.1 HQC

Hamming Quasi-Cyclic (HQC) is a code-based cryptosystem which security relies on the hardness of solving the established syndrome decoding problem. The HQC Key Encapsulation Mechanism (KEM) is created from HQC Public Key Encryption (PKE) using a Fujisaki-Okamoto-like transform called the Hofheinz-Hövelmanns-Kiltz (HHK) transform [HHK17]. The goal of this transform is to reach the IND-CCA2 security, especially by adding a re-encryption phase. HQC ciphertext security stands on the ability of masking a message with random error, so that no one can decode it without the knowledge of the secret key. Thus, the error correction code does not need to be hidden, and any error correcting code can be selected. For HQC-RMRS, authors proposed to use concatenated Reed-Muller (RM) and Reed-Solomon (RS) codes. The main difficulty for HQC, is to prove that the Decryption Failure Rate (DFR), *i.e.*, the decoding failure rate, is smaller than $2^{-\lambda}$ where λ is the security level. This work has been done for the current RMRS version of HQC [AGZ20]. This low DFR on the decoder of HQC implies a property about the DFR of the internal code. Indeed, in most cases, the intermediate codeword between RM decoder and RS decoder is already error-free. The probability of this event occurring is also well studied in [AMAB+17] (page 30, Table 4) and we summarize it in Table 1 below:

Table 1: HQC Decryption Failure Rates from [AMAB+17]

λ	HQC DFR	RM observed DFR
HQC-RMRS-128	$< 2^{-128}$	$2^{-10.96}$
HQC-RMRS-192	$< 2^{-192}$	$2^{-14.39}$
HQC-RMRS-256	$< 2^{-256}$	$2^{-11.48}$

We summarize the HQC-RMRS Framework and the DFR for $\lambda = 128$ bits of security with Figure 1. For HQC-RMRS-128, we then have the following property :

$$\mathbb{P}\left(\tilde{m}\tilde{G} \neq \tilde{m}\tilde{G} + e''\right) < 2^{-10.96} \tag{1}$$

i.e.

$$\mathbb{P}\left(e'' = 0\right) \geq 1 - 2^{-10.96} \tag{2}$$

At the end of the HQC-KEM protocol, $m = m'$ with probability $1 - 2^{\lambda}$, so m will be hashed to derive the shared key. For a KEM, securing this secret value is as important as securing the secret key.

1.1.1 Reed-Solomon Codes

Reed-Solomon Codes (RS) are a sub-class of cyclic codes. These $[n, k, t]$ codes over $\mathbb{F}_q$ are generated using a generator polynomial $g(x) \in \mathbb{F}_q[X]$ of degree $n - k$. The generator polynomial is given as parameter of HQC scheme. Any message $m \in \mathbb{F}_q^k$ can be seen as a polynomial $u(x) = \sum_{i=0}^{k-1} m_i \cdot x^i \in \mathbb{F}_q[X]$. In the reference implementation of HQC, the RS encoding is performed under systematic form, following strategy in [LCM84].

Encoding RS Let $u(x)$ be the polynomial associated to the message m, i.e. $(m_1, \cdots, m_k) = (u_0, \cdots, u_{k-1})$ and $g(x)$ the generator polynomial of the RS code.

$$c(x) = u(x) \times x^{n-k} + \left(u(x) \times x^{n-k} \bmod g(x)\right) \tag{3}$$

In HQC reference implementation [AMAB+], this encoding is performed by Algorithm 1.

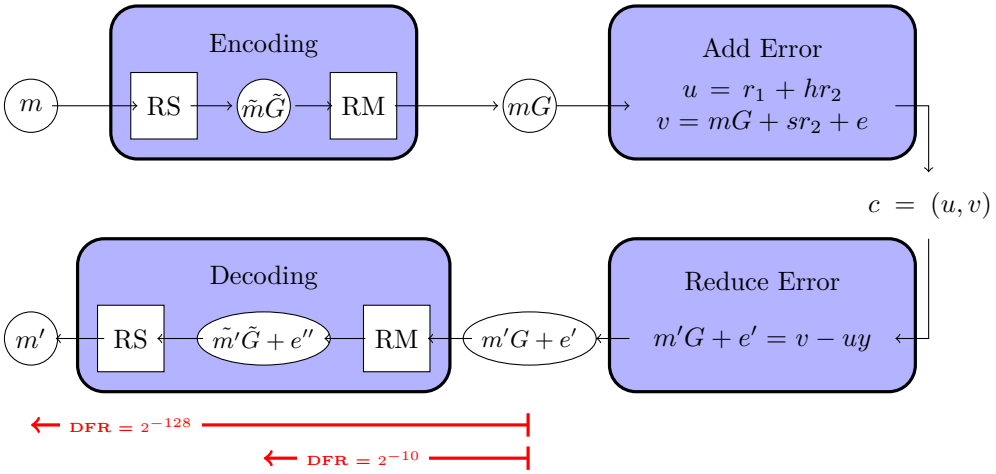

Figure 1: HQC-RMRS Framework and DFR for HQC128

Algorithm 1 HQC Reed-Solomon Encoder from [AMAB+]

Require: parameters: $k, n, \deg_g$
Require: generator polynomial $g \in \mathbb{F}_q^{\mathtt{deg}(g)}$
Require: message $m \in \mathbb{F}_q^k$
Ensure: $c = RS_{\mathtt{encode}}(m) \in \mathbb{F}_q^n$
  Initialize c to 0^n
  for i from 1 to k **do**
    $\Gamma = m[k-i] \oplus c[n-k]$
    for j from 1 to $\mathtt{deg}(g)$ **do**
      $tmp[j] = \mathtt{gf\_mul}\,(\Gamma, g[j])$ ▷ `gf_mul` is the Galois field multiplication
    for l from 2 to $n-k-1$ **do**
      $c[l] = c[l-1] \oplus tmp[l]$
    $C_1 = tmp[1]$
  $c[n-k:n] = m$
  return c

Decoding RS The RS decoder used in HQC follows the theory from [JH04]. The strategy is based on the existence of a unique interpolating polynomial for the received codeword. By solving linear algebra, this polynomial allows decoding up to half the minimum distance of errors. The first operation is the syndrome computation (see Algorithm 2), done with the knowledge of the parity check matrix $H = (h_{i,j})$. As a reminder, the decoder of HQC, and therefore the parity check matrix, are publicly known.

Algorithm 2 Compute Syndromes from HQC RS Decoder from [AMAB$^+$]

Require: parameters: k, n the dimension and length of the code
Require: parity check matric $H \in \mathbb{F}_q^{(n-k,n)}$
Require: codeword $c \in \mathbb{F}_q^n$
Ensure: $s := H^T \cdot c$ the syndrome of c
1: Initialize s to C_1^{n-k}
2: **for** i from 1 to $n-k$ **do**
3: **for** j from 2 to n **do**
4: $s[i] = s[i] \oplus \texttt{gf\_mul}\,(c[j], H[i, j-1])$ ▷ `gf_mul`: Galois field multiplication
5: **return** s

Null syndrome From the low DFR (see Table 1), we know that the input of the RS decoder in HQC is almost always an error-free codeword, which syndrome is zero. For the rest of the paper, we will consider that this codeword is always error-free. As a consequence, after the syndrome computation, the RS decoder will manipulate only zeros; we will not describe the following operations.

Galois field multiplication The main operation during the encoder and the syndrome computation is `gf_mul`, the Galois field multiplication. This algorithm uses a fast multiplication algorithm from [BGTZ08] based on a Fast Fourier Transform (FFT) model. With Algorithm 3, we describe this operation used in [AMAB$^+$], the April 2023 reference implementation of HQC. The `gf_mul` implementation remains the same independently of the HQC selected security level.

Decoding RS list decoders The literature exposes two more efficient strategies to decode RS codes, namely list decoders. This is done by modifying the interpolating polynomial by adding some constraints [JH04]. This strategy allows decoding more error than the classical decoder, but outputs a list of possible decoded messages instead of a single one. Two strategies were discovered by Sudan (S) [Sud00] in 1997 and was improved in 1999 by Guruswami and Sudan (GS) [VG99]. If the weight of the error is below a given threshold τ, depending on the code parameters and the size of the list, the true message belongs to the list. Improved error correcting capabilities for HQC are presented in Table 2.

Table 2: RS parameters and error correcting capability for HQC-λ.

λ	RS k	RS n	RS t	RS S-τ	RS GS-τ
HQC128	16	46	15	15 ($l = 1$)	19 ($l = 13$)
HQC192	24	56	16	16 ($l = 1$)	19 ($l = 9$)
HQC256	32	90	29	29 ($l = 1$)	36 ($l = 15$)

Algorithm 3 Galois field multiplication from [AMAB$^+$]

Require: a, b two elements of $\mathbb{F}_q$
Ensure: $v = a \times b$
  $u[0] = 0$
  $u[1] = b \ \& \ ((1UL << 7) - 1UL)$
  $u[2] = u[1] << 1$
  $u[3] = u[2] \oplus u[1]$
  $g = 0$
  $t_1 = a \ \& \ 3$
  for i from 0 to 4 **do**
    $t_2 = t_1 - i$
    $g = g \oplus (u[i] \ \& \ -(1 - ((t_2 \mid -t_2) >> 31)))$
  $l = g$
  $h = 0$
  for i from 2 to 8 with step 2 **do**
    $g = 0$
    $t_1 = (a >> i) \ \& \ 3$
    for j from 0 to 4 **do**
      $t_2 = t_1 - j$
      $g = g \oplus (u[j] \ \& \ -(1 - ((t_2 \mid -t_2) >> 31)))$
    $l = l \oplus g << i$
    $h = h \oplus g >> (8 - i)$
  $mask = (-((b >> 7) \ \& \ 1))$
  $l = l \oplus ((a << 7) \ \& \ mask)$
  $h = h \oplus ((a >> (1)) \ \& \ mask)$
  $v =$ `gf_reduce` $(l \mid\mid h)$ ▷ Call to Algorithm 4 in Appendix A
  return v

1.2 SASCA with Belief Propagation

Belief Propagation (BP) is a widely used approach in the field of probabilistic graphical models, particularly in Bayesian networks and Markov random fields. It is based on a message-passing algorithm designed to compute marginal probabilities or make inferences about random variables within these models. In the context of SASCA, the graph can be fed with leakage information (*i.e.*, probability distributions) on some intermediate values during the computation of a target algorithm.

Belief propagation is applied on a bipartite graph, called a factor graph, which is composed of two types of nodes: variable nodes that are used to store the probability distributions of the algorithm's intermediate variables, and factor nodes that represent the arithmetical links between them. The process starts with an initialization step where each variable node in the graph receive a former "belief" marginal. This initial belief can come from side-channel leakage, often obtained with a template modeling. Variable nodes with no prior knowledge are initialized with a uniform distribution. Then, the message passing algorithm operates. The message $\mu_{x\to f}$ sent from variable node x to factor node f is defined as follows [KFL01]:

$$\mu_{x\to f}(x) = \prod_{h\in n(x)\setminus\{f\}} \mu_{h\to x}(x) \tag{4}$$

Where $n(x)$ returns the neighbors of x within the factor graph. Additionally, messages sent by a factor f depending on a variable x is computed with the *sum-product* formula depicted as follows:

$$\mu_{f\to x}(x) = \sum_{\sim\{x\}} \left(f(X) \prod_{y\in n(f)\setminus\{x\}} \mu_{y\to f}(y) \right) \tag{5}$$

where X represents the set of variable nodes connected to f and $\sim\{x\}$ expresses the summary notation as defined in [KFL01].

Messages are passed iteratively between nodes in the graph. The algorithm is stopped when the maximum number of iterations is reached or when convergence is reached. The latter allows being more flexible regarding the setup of the maximum number of iterations, at the cost of finding a strategy for detecting convergence. In this paper, we consider that a threshold on the maximal statistical change of all variables' distributions is a satisfying method to detect convergence. In other words, the algorithm stops if distributions of all nodes remains almost constant between two (or more) updates. Eventually, marginal distributions of all variables are extracted as follows:

$$P(x) = \frac{1}{Z} \prod_{f\in n(x)} \mu_{f\to x}(x) \tag{6}$$

with Z being a normalization factor.

The belief propagation algorithm has been proved to be exact on tree-like graphs. In practice, cryptography related graphs often contain cycles, but BP (or loopy-BP in these cases) provides good empirical results. Eventually, several techniques such as message damping and scheduling can be applied when the graph contains cycles: these techniques are not used in this work.

2 Attacker Model

In this paper, we consider an attacker able to perform profiled attacks on HQC decapsulation for shared key recovery. This implies that the adversary has access on a fully controlled

clone of the real target device for the profiling phase. For simplification purposes, we run both profiling and attack procedures on the same physical device: the complexity of template portability does not fall under the scope of this paper. Throughout this work, we suppose the attacker to be able to craft templates from the `gf_mul` operation only. Hence, we suppose that the attacker has the ability to isolate an ordered sequence of `gf_mul` computations within a wider routine, such as the RS decoder or encoder. We believe that this task can be conducted thanks to pattern matching techniques, and is then eluded from our study. Eventually, as all attacks presented in this paper target the HQC shared key, the attacker does not have the ability to increase the Signal-to-Noise Ratio (SNR) with techniques requiring side-channel measurement of several HQC decapsulation instances (such as trace averaging).

3 Single Template Attack

In this section, we implement an attack aiming at recovering all the codeword bytes of the error-free codeword by using only one template. In a second phase, this codeword can be decoded to deduce the shared key computed at the end of the key exchange. Finally, we describe how the decoder structure allows coping with eventual template mispredictions and obtain high attack success rates.

3.1 Experimental Setup

We acquired traces with a "Langer Near Field" electromagnetic probe using a RT02024 Rhode-Schwarz oscilloscope with a sample rate of 1 GHz. The Galois field multiplication `gf_mul` has been extracted from the April 2023 reference implementation of HQC [AMAB+] following Algorithm 3. We selected the STM32F407 as our target board. We compiled the code with $-O3$ optimization, surrounded by a GPIO based trigger. Assembly code of this function can be seen in Appendix B. This set-up leads to a computation time of $1.3\mu s$ and traces of 1300 points, see Figure 2 for the average acquired trace. These small-sized traces allowed us to perform our attack on the full length of the traces, without selecting points or areas of interest. In total, we acquired an amount of 500000 traces for randomly sampled inputs.

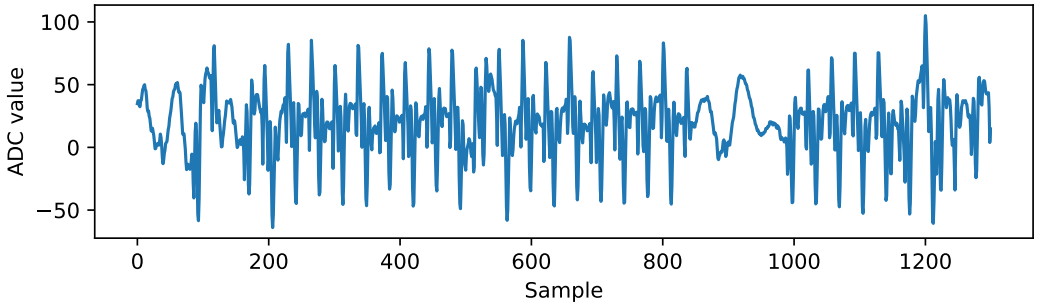

Figure 2: Mean trace of `gf_mul` function execution.

3.2 Templates on Galois Field Multiplication

Before prior templating phase, we conduct a leakage assessment on the three 8-bit variables involved in `gf_mul`: the two multiplication operands as well as the output. We make the assumption of a linear leakage model and rely on a Linear Regression Analysis (LRA).

Namely, for a side-channel measurement x_i and an 8-bit variable y_i, we express the leakage as:

$$x_i = \beta_0 + \sum_{j=1}^{8} \beta_j \cdot y_{i,j} + \epsilon \tag{7}$$

Given a set of n training samples $(x_i)_{1 \leq i \leq n}$ (i.e., n traces) and under Gaussian noise ϵ, there exists a unique solution to this system $\tilde{\beta} = \left(\tilde{\beta}_0, \cdots, \tilde{\beta}_8\right)$, *i.e.*, an estimation of the parameters $\beta = (\beta_0, \cdots, \beta_8)$, which minimizes the residual sum of squares defined as:

$$\begin{aligned} RSS &= \sum_{i=1}^{n} (x_i - \tilde{x}_i)^2 \\ &= \sum_{i=1}^{n} \left(x_i - \left(\tilde{\beta}_0 + \sum_{j=1}^{8} \tilde{\beta}_j \cdot y_{i,j} \right) \right)^2 \end{aligned} \tag{8}$$

The accuracy of the model can be measured through the coefficient of determination, denoted R^2, which is computed as follows:

$$R^2 = 1 - \frac{\sum_{i=1}^{n} (x_i - \tilde{x}_i)^2}{\sum_{i=1}^{n} (x_i - E(x))^2} \tag{9}$$

Note that this metric needs to be computed in a univariate way (*i.e.*, for each time sample). Coefficients of determination corresponding to the three targeted variables are displayed in Figure 3.

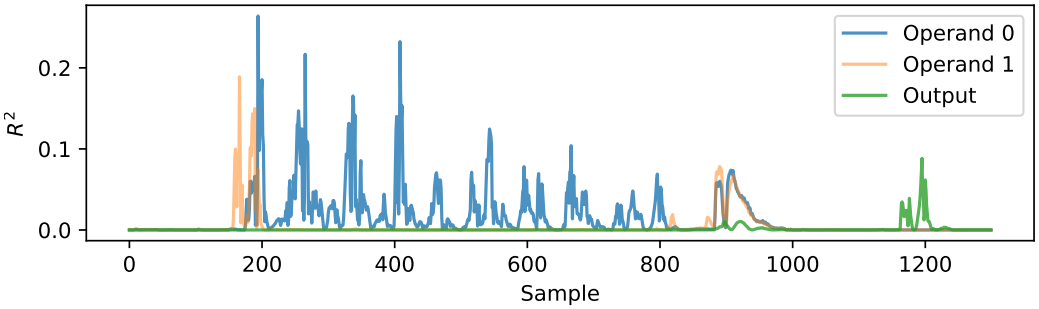

Figure 3: Coefficients of determination computed for both inputs and the output of the Galois field multiplication.

By observing the LRA output in Figure 3 we can observe that (i) the leakage of the first operand is both important and spread along the computation of `gf_mul`: this can be explained by the several logical operations perform on this operand, (ii) the leakage corresponding to the second operand is less important and (iii) the output of `gf_mul` computation is leaking at the end of the function, probably when it is stored in main memory (see Appendix B).

Then, we can conduct a template attack aiming at making predictions on these three variables from the side-channel measurements. We use Fisher's Linear Discriminant Analysis (LDA) as our classifier. For each variable, we evaluate the accuracy of two models, respectively predicting the variable value or its Hamming weight. The validation accuracy of each model is evaluated on datasets of different sizes segmented into 90% training and 10% validation traces. The results of these experiments are displayed in Figure 4. Several observations can be made:

- the value of the first operand can be predicted with a 93.89% accuracy considering a dataset of 300000 traces,
- the value template attacks on second operand and output value do not provide predictions significantly better than random guesses,
- the Hamming weights of the output and second operand give satisfactory results, more informative than a random guess.

Template accuracies for 300000 training traces are described in Table 3 in Section 8. We can conclude that the high number of logical operations performed on the first `gf_mul` operand (see Algorithm 3) is beneficial from a template attacker's perspective. As a reminder, during the computation of the RS syndromes (see Algorithm 2), the message, which is the sensitive data of this computation, is used as the first operand of the multiplication which is the one that leaks the most. Moreover, the storing of `gf_mul`'s output in main memory allows an attacker to reach exploitable template accuracies.

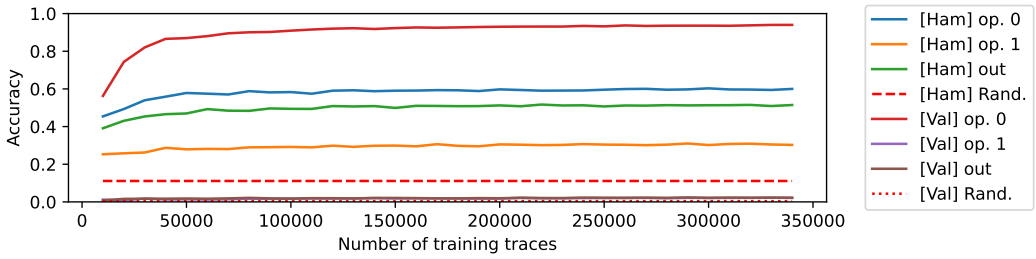

Figure 4: Accuracy on each variable for both value and Hamming weight templates depending on the number of training traces.

3.3 Building Prediction Matrices

Designing a simulation that matches the predictions of a template attack on a real target is a hard task. Indeed, the outputs of the templates depend on several factors. For instance, the hardware components involved in the attack, such as the target board and the measurement chain, and the experimental setup conditions (*e.g.*, EM probe positioning, temperature *etc.*) have an impact on the template accuracy. The choice of a classifier, as well as its exploitation of multivariate leakage, also have a considerable impact on the template's properties.

For these reasons, our real-case scenario attacks are performed thanks to prediction matrices. The latter contain a set on 100000 independent probability distributions predicted by the models displayed in Subsection 3.2, along with the corresponding true labels. The advantages of such prediction matrices are twofold. Firstly, randomly sampled elements from a prediction matrix can be seen as a real template prediction: this can be used in a simulation context to test the robustness of the attacks, that can easily be ran a high number on times. Secondly, as all attacks presented in this paper exploit the leakages of the `gf_mul` operation, the use of prediction matrices allows deriving attacks on several functions and countermeasure scenarios.

In our particular case, we stress that claiming that the use of prediction matrices is comparable to a real case scenario attack highly depends on the ability for the attacker to detect the `gf_mul` routines in wider side-channel traces. We believe that this assumption is reasonable within the attacker model we consider in this paper.

3.4 Combine and Conquer

In the current reference implementation of HQC [AMAB+], `gf_mul` always manipulates the codeword bytes under the first operand. This choice allows an attacker to exploit the high first operand template accuracy that we denote p. For each codeword byte, $n-k$ independent trials can be combined by multiplying marginal distributions. To simplify the mathematical analysis of such an approach, we model the latter as a majority voting strategy. The goal is to provide a lower bound for the success rate of this attack.

Majority voting The probability that the good hypothesis is ranked first by the classifier can be seen as the result of a Bernoulli distribution with parameter p. Given that trials are independent, they can be combined into a binomial distribution with parameters $n-k$ and p. Let's denote by X the random variable following this distribution for a codeword byte. One can observe that C_1, the first codeword byte, is not manipulated with `gf_mul`, and hence cannot be recovered with our template attack. For any other codeword byte, the majority voting is a success if and only if $X > \lfloor \frac{n}{2} \rfloor$. Furthermore, all codeword bytes are independent, which results into the following success probability:

$$\mathbb{P}\left(\texttt{success}_{\setminus C_1}\right) = \mathbb{P}\left(X > \left\lfloor \frac{n}{2} \right\rfloor\right)^{n-1}, X \rightsquigarrow \mathcal{B}(n-k, p) \tag{10}$$

3.5 Re-Decoding Strategy

In this paper, we apply the strategy from [GLG22b] that consists in re-decoding the recovered codeword which provides several advantages: (i) re-decoding allows correcting templates mistakes or inaccuracies, (ii) this allows at recovering the value of C_1 which cannot be found by a template results and (iii) the attacker gains additional flexibility regarding the accuracy of the template. Recall from Section 1.1.1 that one is able to decode up to τ errors for a Reed-Solomon code with such parameters. Note that one error slot is already taken by C_1, hence the probability of success of the attack becomes:

$$\mathbb{P}\left(\texttt{success}\right) = \sum_{i=0}^{\tau-1} \mathbb{P}\left(X > \left\lfloor \frac{n}{2} \right\rfloor\right)^{n-i-1} \cdot \mathbb{P}\left(X \leq \left\lfloor \frac{n}{2} \right\rfloor\right)^{i}, X \rightsquigarrow \mathcal{B}(n-k, p) \tag{11}$$

3.6 Practical Attack

Targeting all security levels This template attack can also be conducted for HQC higher security levels. In fact, the `gf_mul` function is exactly the same, independently of the selected security level, allowing us to re-use templates (see Subsection 3.2). Parameters from Table 2 show that n, the number of codeword bytes to be recovered, increases with the security level. But, at the same time, $n-k$, the number of independent trials, also increases, giving more independent information about each codeword byte.

Results We observe an accuracy of $p = 0.9389$ on the first operand with 300000 training traces and a single attack trace (see Figure 4). Considering this probability in equations 10 and 11, we obtain success rates greater than 0.9999 with or without the re-decoding strategy for all HQC security levels.

Discussion From Equation 11, we compute the minimum value of p such that the success of the attack stays beyond 0.9. It follows that a template accuracy of $p_{min} = 0.7262$ is enough to succeed in the attack for HQC128. HQC192 and HQC256 require minimal template accuracy being respectively $p_{min} = 0.7250$ and $p_{min} = 0.6834$. Since the minimal

required accuracy is lower for each security level than what we obtained in practice, we could consider attacking targets with higher noise level.

This first attack is based on an unfortunate choice of operand order for the multiplication in the reference implementation of HQC [AMAB+]. We can reasonably assume that, from the results presented herein, a informed developer will make the choice to swap first and second operands. This allows manipulating sensitive data under the operand that leaks the least. Given that this multiplication operation is commutative, swapping operands does not imply computational overhead.

4 SASCA on Reed-Solomon Decoder

After the swap of operands, we are not able to realize the attack from Section 3 against the sensitive data, which is now "hidden" behind the second operand. Moreover, the first operand, which value can be templated with high accuracy, now holds the content of the parity check matrix which is already publicly known. Nevertheless, Figure 4 and Table 3 show that the Hamming weight of `gf_mul`'s second input and output can be templated with high accuracy. This information is gathered into a factor graph.

4.1 Reed-Solomon Decoder Graph

We build a factor graph representing the RS syndrome computation (see Figure 5). This graph allows seeing the relation between each intermediate value used during the computation. In a normal use of a decoder, the output syndrome gives information about the random error added to the codeword. But here, we consider the RS syndrome as zero (see Subsection 1.1) allowing removing the lower part of the graph. This construction ends up with $n-1$ windows, each representing an independent tree-like graph and benefit from the BP convergence proof in such graph topology. We recall that re-decoding strategy is available for the attacker, allowing them to recover C_1, the first codeword byte which is outside all windows.

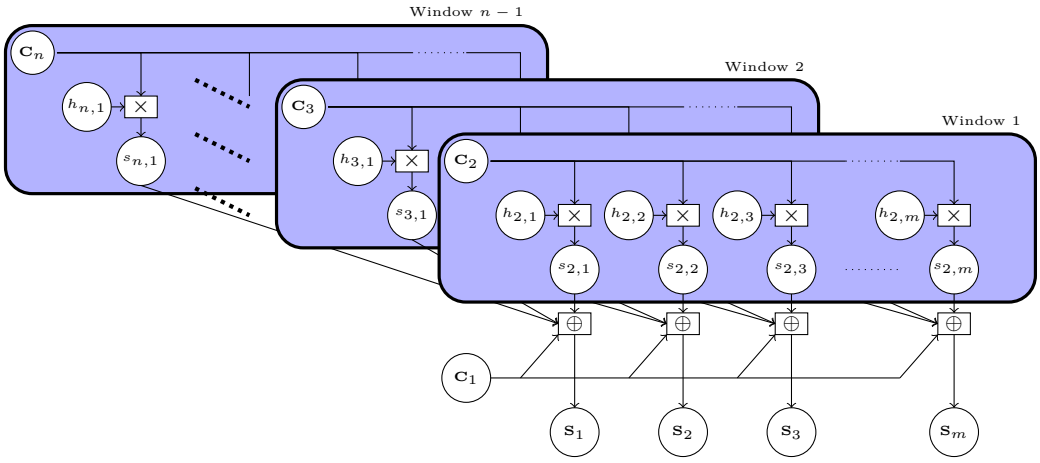

Figure 5: Reed-Solomon decoder (syndrome computation) factor graph (with $m = n - k$).

Building `gf_mul` sub-graph The main sub-operation performed during the RS syndrome computation is `gf_mul`, the multiplication in the Galois field $\mathbb{F}_{2^8}$. This operation can

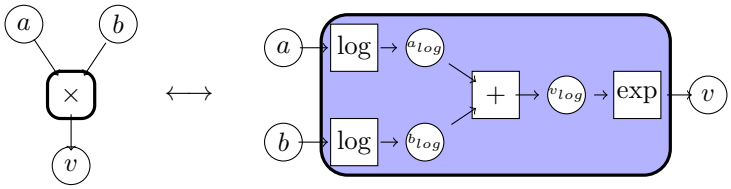

Figure 6: Galois field multiplication sub-graph. Factors are denoted with a square and variables with a circle.

be performed using a fast multiplication based on the Fast Fourier Transform (see Algorithm 3) [BGTZ08], which is the choice of the HQC authors since April 2023 in the reference implementation. However, this calculation can be done differently, using the logarithm representation of each element. $v := a \times b = \alpha^{\texttt{log}(a)} \times \alpha^{\texttt{log}(b)} = \alpha^{(\texttt{log}(a)+\texttt{log}(b))\%n}$, where α is a primitive element of the Galois Field. After this transformation, if the `log` and `exp` transformation (stored with precomputed tables in practice) are known, the multiplication can be computed by simple addition and modular reduction. This approach allows us to optimize the computations of factor messages (see Figure 6). Namely, lookup tables are used to compute logarithm, exponentiation and modular reduction factor operations. The addition factor is implemented with a convolution, which can benefit from a FFT depending on the size of the variables' domain.

4.2 Simulating Hamming Weight Leakages

As a first step, we perform simulations on the decoder graph. We initiated the marginal probability of the second operand with the high accuracy value template results from prediction matrices (see Subsection 3.3). This operand gives information about the parity check matrix elements. We also initiated the marginal probabilities of the outputs of all `gf_mul` computation from a Hamming weight leakage model with a Gaussian noise. Hence, for a side channel trace x resulting from the manipulation of a `gf_mul` output v, we have:

$$x = \alpha \cdot \texttt{HW}(v) + \mathcal{N}(\beta, \sigma^2) \tag{12}$$

With this leakage model, we can simulate the output of a perfect template classifier with the following equations:

$$\mathbb{P}\left(\texttt{guess} \ = v \mid \texttt{label} = l\right) = \mathbb{P}(X = v), X \rightsquigarrow \mathcal{N}(l, \sigma^2) \tag{13}$$

Simulation results Figure 7 shows the success rate of the attack when increasing the standard deviation value σ step by step, and performing the attack 400 times for each of them. Up to a standard deviation of 2, we have a success rate of 1 for security levels 128 and 192. For the highest security level of HQC, we can almost reach a noise level of $\sigma = 3$ without loss of accuracy.

Discussion We observe that the success rate for HQC192 is lower than the one for HQC128. Indeed, the number of bytes to recover is larger (56 instead of 46), the number of independent trials remains almost the same (32 instead of 30) and the error correction capability is the same (*i.e*, 19). This makes the attacks more difficult to conduct considering Equation 11.

We expected the attack on HQC256 to have a success rate greater than for HQC128. Indeed, the number of codeword bytes to recover is way larger for this security level ($n = 90$ instead of 46 or 56), but the number of independent trials is also bigger ($n - k = 58$ instead

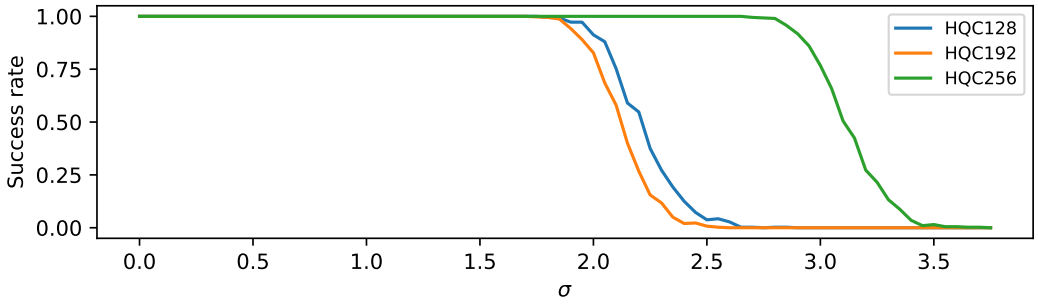

Figure 7: Simulated success rate of SASCA on the decoder, with re-decoding strategy, depending on the selected security level of HQC.

of 30 or 32), and the error correction capability increases (36 instead of 19). The attack needs to find twice as many codeword bytes, but has twice as many independent leaks on each of them, and is ultimately helped by a strong correction capability.

In order to lower the success rate of our attack, the parameters of HQC should be chosen with aim at reducing the probability of success given in Equation 11. This can be done by lowering the value of the error correction capability. This option seems to be in contradiction with the expected choice for a code $\mathcal{C}$ in HQC in order to reduce the scheme ciphertext and key sizes. Another option can be to increase n, the number of codeword bytes to recover, without increasing $n-k$, the number of independent leaks on each of them. This can be done by selecting a code with higher rates $R = \frac{k}{n}$.

5 Masking Countermeasure

Protecting the decoder of HQC against SCA can be performed by using codeword masking [MSS13]. This masking strategy allows creating a mask for the decoder using an encoder. Instead of decoding $c + e$ into m, we start by randomly sampling a message mask m'. This message mask is encoded into c', the codeword mask. Then the decoder algorithm is applied on $c + c' + e$, masking the sensitive data c, returning $m + m'$ due to the linearity of the involved code. The true result m is recovered by subtracting the message mask m'. Since the encoder is a fast operation in front of the decoder, codeword masking allows reducing the overhead of the countermeasure.

Given that the countermeasure is not the repetition of the same operation, codeword masking requires a further study about the encoding algorithm. Consequently, an attack targeting a masked implementation of HQC can be performed in two steps: (i) attacking the decoder to recover $c + c'$, the masked shared key and (ii) attacking the encoder to recover the mask c'. In this scenario, the success rate is the product of both success rates of these two points. The first point is addressed in Section 4. In this section, we describe a SASCA approach against the RS encoder, addressing the second point.

5.1 Reed-Solomon Encoder Graph

We are able to represent the RS encoder (see Algorithm 1) as a graph (see Figure 8). As well as the decoder, the main operation during the encoder is `gf_mul`, the Galois field multiplication. Moreover, this encoder algorithm requires the knowledge of g, a generator polynomial, publicly known as a parameter of HQC. This prior knowledge can be implemented into the factor graph. Finally, the attack also aims at recovering the RS codeword bytes, allowing applying the re-decoding strategy (see Subsection 3.5).

We re-used the same template results from Subsection 3.3 to perform practical attacks. Simulations follow theory from Hamming weight leakage model from Subsection 4.2 which results are in Figure 9.

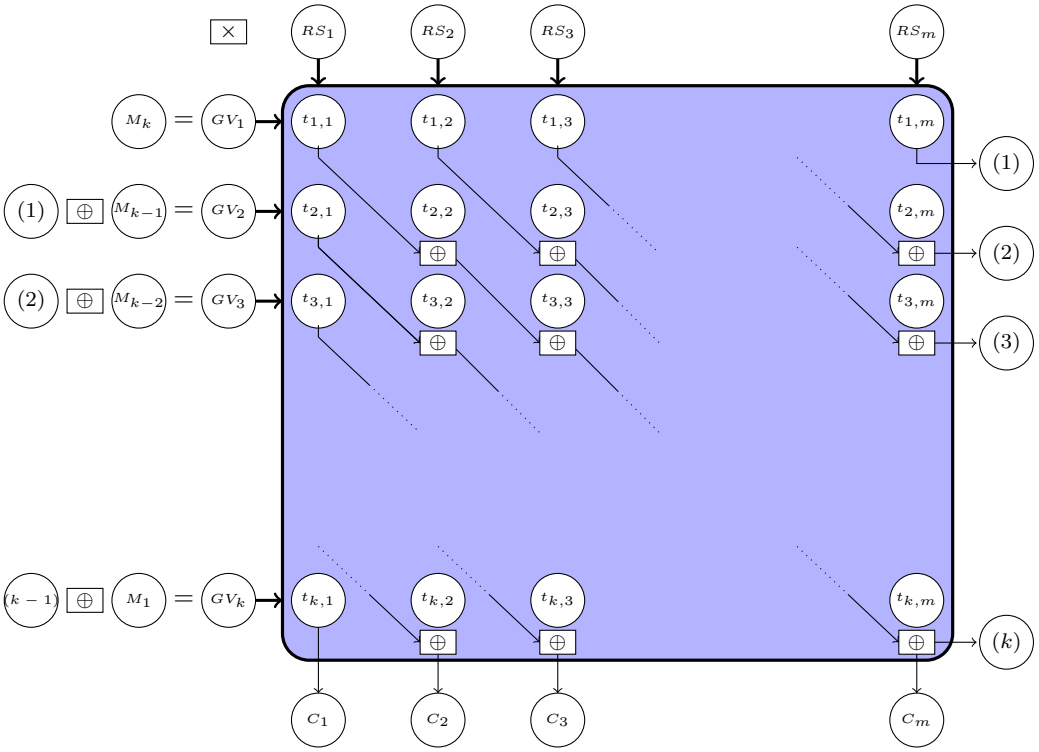

Figure 8: RS Encoder seen as a Graph (with $m = n - k$).

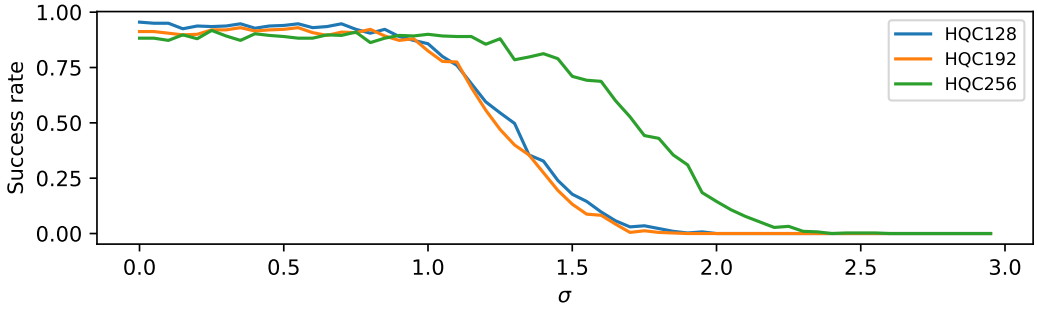

Figure 9: Success rates of SASCA against the Reed-Solomon encoder, with re-decoding strategy, depending on the selected security level of HQC.

Results Simulation results displayed in Figure 9 show the attack on the encoder is more sensitive to noise than the decoder's (see Figure 7). We claim that, in practice, the masked decoder is not secure since we are still able to recover the mask with a probability of 0.7625, 0.6575 and 0.8075 for HQC128, HQC192 and HQC256 respectively.

Discussion The sensitivity of this attack to noise can potentially be explained by the sparse relations between intermediate values in the encoder graph, as well as cycles within the latter. Future work can focus on optimization techniques such as damping or message scheduling. Still, higher success rates for HQC256 are reported, both in simulations and real case scenario.

High-order masking strategy By generating N random masks and adding them together, one can generate high-order masking. Each one of the N masks must be independently recovered to succeed in the attack, which occurs with probability $p_N = p^N$, with p the probability of recovering a single mask. It follows that reducing the probability of success under 0.01 requires to compute 17, 11 or 22 independent masks for HQC128, HQC192 and HQC256 respectively. This approach doesn't seem to be effective due to the additional overhead it incurs.

6 Shuffling Countermeasures

The NTT from Kyber [BDK+18] was already targeted by SASCA like strategies in [PPM17, PP19, HHP+21]. Two shuffling countermeasures, coarse-full-shuffling and fine shuffling [RPBC20] were identified to protect the NTT against SASCA. The fine shuffling aims at shuffling the order of NTT inputs and outputs, randomly selecting one of the 4 combinations for each call. This strategy prevents an attacker from labeling the observed leakages. The coarse shuffling consists in shuffling the elements of the inner loop, independently within each layer. These shuffling strategies can be adapted to protect HQC. Indeed, the layer of the NTT behaves like the windows of the RS decoder. The coarse shuffling can be used to shuffle elements order within a window (see Figure 5). The fine shuffling can be used to shuffle the inputs of `gf_mul`, since the output is unique, the number of combinations is just 2.

We can also deduct novel shuffling methods for HQC. A possibility is to compute each window in a random order. This strategy is useless for the NTT since all layers perform the exact same operation. However, for the RS decoder, the windows are perfectly independent and can then be performed in a random order. We call this countermeasure window shuffling. Finally, all `gf_mul` operations being independent during the computation, they can be performed in a fully random order, following ideas from [ATT+18].

In this section, we describe and analyze the security of these shuffling countermeasures. For the study of shuffling countermeasures from a side-channel perspective, we emphasize the importance for the attacker to possess a fully controlled device for the profiling phase. Indeed, either the knowledge of the shuffling or the possibility to isolate a single known `gf_mul` operations is mandatory to craft templates.

6.1 Fine Shuffling

Under a fine shuffling strategy, the sensitive data (*i.e.*, the codeword byte) is manipulated under the first operand one out of two times in average. We re-use a majority voting strategy from Subsection 3.4 to exploit the high first operand leakage. We consider that the output of the classifier is a random value when the sensitive data is hidden behind the second operand. This hypothesis is a worse scenario than what we do observe in practice (see Figure 4). Indeed, the leakage on the values from the parity check matrix could help for a more refined analysis. However, if the probability that the good hypothesis is ranked first by the classifier is high enough, the majority voting will succeed.

Discussion Using the fine shuffling strategy goes against the desire to hide the sensitive data under the operand that leaks the least, as discussed in Section 3.

6.2 Coarse Shuffling

The coarse shuffling strategy aims at shuffling the operations' order performed in each window. The selected shuffling can be changed for each window, ensuring a better security level. The sequence of operations does not impact the graph construction or the path to convergence for the target codeword. This assertion is true since the $n-1$ windows are independent sub-graphs (see Section 4). We recall that the value of C_1 is recovered with the final re-decoding strategy (see Subsection 3.5). Consequently, this case matches our prior setup of belief propagation, giving the same results as the previous decoder attack (see Section 4).

6.3 Window Shuffling

Shuffling windows allows interchanging the order of codeword bytes computations. In such a case, even if we are able to converge with a BP attack, recovered codeword bytes are shuffled and the attack does not succeed unless the permutation is reversed. Here, we apply the same attack strategy, independently of the considered swap order. Indeed, the first step is to the run the belief propagation as presented in Section 4. This step produces marginal probabilities on each intermediate value. Previous results show that the values of the codeword bytes are successfully recovered, independently of the presence of a shuffling. Consequently the difficulty of attacking this shuffle remains in inverting the permutation.

Inverting codeword bytes permutation The parity check matrix $B = (b_{i,j})_{\substack{1\leq i\leq k\\ 1\leq j\leq n}}$ can be transformed into a Dirac probability distribution under matrix T of size $k \times n \times 256$:

$$T[i,j,l] = \begin{cases} 1 & \text{if } b_{i,j} = l \\ 0 & \text{otherwise} \end{cases} \tag{14}$$

We know that the lines of the parity check matrix has been shuffled by the window shuffling, but in each line, elements kept their original arrangement. After the first BP phase, we obtain a shuffled estimation of T, denoted $\tilde{T}$, that holds the marginals of each variable representing B. More formally, if L represents a side-channel measurement:

$$\tilde{T}[i,j,l] = \mathbb{P}\left(b_{i,j} = l \mid L\right) \tag{15}$$

The idea is to reassign the lines $\tilde{T}$ in order to minimize a distance with T. To do so, we compute the matrix D such that:

$$D[i,i'] = \sum_{j=1}^{256} \mathtt{d}(\tilde{T}[i,j],\ T[i',j]) \tag{16}$$

where $\mathtt{d}$ is an arbitrary distance function. Inverting the window shuffling is equivalent to select k elements from the matrix D. Exactly one element per row and one element per column such that the sum of these elements is minimal. The location of these selected elements gives the assignment between $\tilde{T}$ and T lines. This problem is an instance of the assignment problem, for which an optimal solver is known.

Assignment problem The assignment problem is a classic optimization problem in the field of operations research and linear programming. It involves finding the optimal assignment of a set of tasks to a set of agents (or workers) in such a way that the total cost or time required to complete the tasks is minimized, or conversely, the total profit or utility is maximized. Each task must be assigned to exactly one agent, and each agent can only be assigned to one task.

Hungarian algorithm The Hungarian algorithm is an efficient method for solving the assignment problem, especially when the problem involves equal numbers of tasks and agents. It was developed by Harold Kuhn [Kuh55] in the 1950s and later refined by James Munkres [Mun57]. We applied it considering $\tilde{T}$ resulting from simulated leakage to study the behavior of the algorithm with noise. Several distance metrics have been evaluated for Equation 16: the L_1 distance[1] presented the best results. After the Hungarian method, we know the value of the second operand with precision. Now (i) either the marginals on the codeword are already satisfactory to foresee a successful re-decode or (ii) the attacker can inject the newly learned information into the graph to converge towards more accurate results.

6.4 Full Shuffling

A stronger shuffling is introduced by combining ideas from window shuffling and coarse shuffling. These two shuffling method can be applied independently as in [GLG22b], but this may lead to de-shuffling attacks. Therefore, they can be cross-used, by totally randomizing the order of `gf_mul` computations. This strategy follows an idea from [ATT+18] and aims at increasing the combinatorial complexity for the attacker. We call this strategy "full shuffling", which algorithm is presented in Appendix C. The overhead of this countermeasure is the cost of shuffling a list of size $n \times (n-k)$.

Complexity of full shuffling inversion Let's suppose that we are able to recover, with a BP attack or other, the exact value of the second operand, coming from the parity check matrix. Given this information, we want to invert the shuffling of these elements. However, the size of the matrix is much larger than the size of the Galois field, leading to a large redundancy. Consequently, it is impossible to un-shuffle without testing all possibilities for the redundant elements. Given the parity check matrix, one is able to compute this number of permutations for all the security levels of HQC. Note that this number increases with the size of the matrix, therefore with the security level, since the Galois field remains the same. This number of permutations is respectively 2^{504}, 2^{614} and 2^{1030} for the three security levels of HQC. This number being larger than the security level, we conclude that inverting the shuffling is not achievable with the strategy presented in Section 6. This leads us to believe that full shuffling is an effective countermeasure against our attack.

Discussion One strategy to break the full shuffling countermeasure could be to attack the random permutation generation. As this attack is outside the scope of our study, we leave it to future work.

7 Decapsulation attack

The HHK transform used for HQC, generally the Fujisaki-Okamoto (FO) transform, involved a re-encryption part during the decapsulation. Thus, a decoded shared key is also re-encoded during the re-encryption. This additional step allows exploiting side-channel leakages from both a decoder and an encoder during the same decapsulation process.

Combining RS decoder and encoder graphs We are able to build a double graph, creating a connection between encoder and decoder graphs. Indeed, these two graphs share the same codeword bytes variable nodes, which hence can be merged. We follow simulation strategy from Subsection 4.2 and display the results in Figure 10. We show

[1]The L_1 distance (also called taxicab or Manhattan distance) between two vectors $\mathbf{x} = (x_1, \cdots, x_n)$ and $\mathbf{y} = (y_1, \cdots, y_n)$ of same length is given by $\mathtt{d}_{L_1}(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} |x_i - y_i|$.

that we are able to reach higher noise levels than any previous attacks in this paper, this for all HQC security levels.

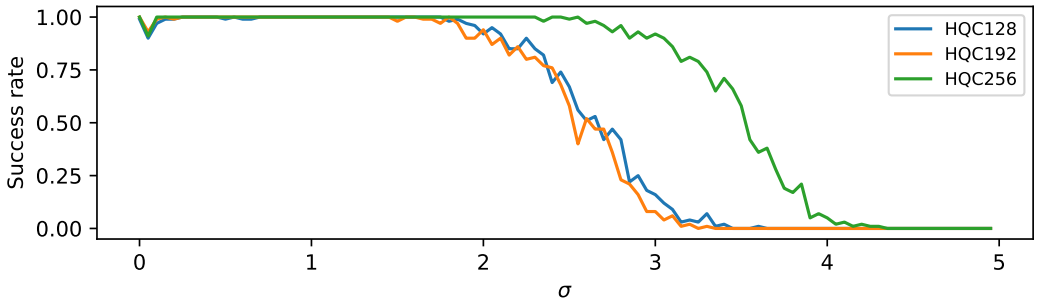

Figure 10: Success rate of SASCA on the decapsulation (decoder + encoder combined), with re-decoding strategy, depending on the selected security level of HQC.

Countermeasure This combined attack, exploiting leakage redundancy from the re-encryption, is a threat to the security of HQC. Then, finding a countermeasure both for the encoder and decoder is required. The current RS encoder algorithm (see Algorithm 1) is implemented with a polynomial division. Protecting this encoder with a shuffling strategy is a hard task, since the carry propagation implies that several `gf_mul` operations depend on the result of previous ones. This constraint makes it impossible to perform them in a fully random order. By considering another encoding algorithm with less operation dependency, such as a classical matrix-vector multiplication, one can apply the full shuffling countermeasure. As mentioned in Subsection 6.4, the combinatorial complexity provided by the full shuffling is sufficient to prevent from our attacks, protecting both the encoder and decoder.

8 Practical Results

For all our attacks, we used the setup presented in Subsection 3.1. For each security level, attacks have been repeated 400 times. Success rates are summarized in Table 3. We stress that our attacks are a threat for HQC and efficient countermeasures such as full shuffling on the decoder and encoder must be applied.

Table 3: Hamming and value tempaltes accuracies on `gf_mul` and success rates of attacks on STM32F407. Each attack has been performed 400 times.

	HQC128	HQC192	HQC256
`gf_mul` templates accuracies: operand 0 (value) operand 1 (Hamming) output (Hamming)	 **0.9389** (value) 0.0211 (value) 0.0221 (value)	 0.5929 (Hamming) **0.3035** (Hamming) **0.5178** (Hamming)	
Decoder	1	1	1
Fine shuffling	1	1	1
Coarse shuffling	1	1	1
Window Shuffling	1	1	1
Encoder (Masking)	0.7625	0.6575	0.8075
Encoder + Decoder (Decaps)	1	1	1

9 Conclusion and Further Work

In this paper, we present new shared key recovery attacks on the code-based PQC NIST contest candidate HQC. Depending on HQC implementation choices, our attacks can either be a classical template attack or rely on Soft Analytical Side-Channel Attack (SASCA) based on Belief Propagation (BP) theory. All the practical attacks presented in this paper are performed within a few minutes on a STM32F407 target running the reference implementation of HQC [AMAB+]. This work takes advantage of the inner structure and properties of code-based cryptography to mount practical shared key recovery attacks.

- We demonstrate practical attacks against the Reed-Solomon (RS) decoder of HQC. Precisely, we exploit physical leakages during Galois field multiplication, a cornerstone operation of the RS logic, and model intermediate variables' dependencies within a factor graph. We simulated this attack with Hamming weight leakage model and showed that the success rate stays high (superior to 0.9) up to $\sigma = 2$ and even $\sigma = 3$ for the highest HQC security level (see Figure 7). In practice, this attack has a success rate of 1 (see Table 3).

- We perform the same analysis against a version of HQC protected with codeword masking. Specifically, the robustness of the RS encoder against SASCA is studied. It emerges that the encoder attack is more sensitive to noise. Simulation results are depicted in Figure 9 with good accuracies up to $\sigma = 1$, and practical accuracy are shown in Table 3. We emphasize that the success rate in a real case attack scenario is enough to threaten the security of the scheme; masking is not an efficient countermeasure to protect HQC against SASCA on a STM32F407.

- We analyze the security of several RS decoder shuffling countermeasures against our attacks. We demonstrate insufficient protection brought by shuffling countermeasures adapted from the Kyber-related literature. Namely, we reach perfect accuracy on a real case attack scenario. We present the full shuffling strategy which provides satisfactory additional combinatorial complexity to the attacks proposed in this paper. We believe that RS decoder full shuffling strategy is an interesting lightweight countermeasure that could possibly thwart other attacks.

- Finally, by exploiting the Fujisaki-Okamoto (FO) transform, an attacker can combine encoder and decoder leakages by merging both factor graphs, for successful shared key recovery on devices with higher noise levels (see Figure 10). The combined attack exploiting the redundancy leakage from the re-encryption is a potential threat for any FO-like scheme. We show that changing the HQC encoding strategy allows protecting both encoder and decoder with full shuffling.

The analysis of HQC's internal Reed-Solomon through the lens of a side-channel attacker leads to several intuitions about further work. Firstly, as all our attacks exploit the Galois field multiplication, we believe that protecting the latter operation is a promising path towards efficient countermeasures. An option could be to implement a gadget [BBE+18] for `gf_mul`, ensuring security for RS operations under a given attacker model. Secondly, the full shuffling algorithm must be carefully selected, especially the random generator, to prevent permutation recovery attacks. Finally, the resilience of other PQC schemes built with the FO transform needs to be evaluated against SASCA approaches analogous to the decapsulation attack presented in this paper. Attacks combining the redundancy of leakages created by the re-encryption could be a threat for FO schemes.

References

[AAC+22] Gorjan Alagic, Daniel Apon, David Cooper, Quynh Dang, Thinh Dang, John Kelsey, Jacob Lichtinger, Carl Miller, Dustin Moody, Rene Peralta, et al. Status report on the third round of the nist post-quantum cryptography standardization process. *US Department of Commerce, NIST*, 2022.

[ABB+17] Nicolas Aragon, Paulo Barreto, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Shay Gueron, Tim Guneysu, Carlos Aguilar Melchor, et al. BIKE: bit flipping key encapsulation. 2017.

[AGZ20] Nicolas Aragon, Philippe Gaborit, and Gilles Zémor. HQC-RMRS, an instantiation of the HQC encryption framework with a more efficient auxiliary error-correcting code. *arXiv preprint arXiv:2005.10741*, 2020.

[AMAB+] Carlos Aguilar-Melchor, Nicolas Aragon, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Edoardo Persichetti, and Gilles Zémor. HQC reference implementation.

[AMAB+17] Carlos Aguilar-Melchor, Nicolas Aragon, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Edoardo Persichetti, and Gilles Zémor. Hamming quasi-cyclic (HQC). 2017.

[ATT+18] Aydin Aysu, Youssef Tobah, Mohit Tiwari, Andreas Gerstlauer, and Michael Orshansky. Horizontal side-channel vulnerabilities of post-quantum key exchange protocols. In *2018 IEEE international symposium on hardware oriented security and trust (HOST)*, pages 81–88. IEEE, 2018.

[BBE+18] Gilles Barthe, Sonia Belaïd, Thomas Espitau, Pierre-Alain Fouque, Benjamin Grégoire, Mélissa Rossi, and Mehdi Tibouchi. Masking the glp lattice-based signature scheme at any order. In *Advances in Cryptology–EUROCRYPT 2018: 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29-May 3, 2018 Proceedings, Part II 37*, pages 354–384. Springer, 2018.

[BCL+] Daniel J Bernstein, Tung Chou, Tanja Lange, Ingo von Maurich, Rafael Misoczki, Ruben Niederhagen, Edoardo Persichetti, Christiane Peters, Peter Schwabe, Nicolas Sendrier, et al. Classic mceliece: conservative code-based cryptography.

[BDK+18] Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. Crystals-kyber: a cca-secure module-lattice-based kem. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 353–367. IEEE, 2018.

[BGTZ08] Richard P Brent, Pierrick Gaudry, Emmanuel Thomé, and Paul Zimmermann. Faster multiplication in gf (2)[x]. In *Algorithmic Number Theory: 8th International Symposium, ANTS-VIII Banff, Canada, May 17-22, 2008 Proceedings 8*, pages 153–166. Springer, 2008.

[CCJ+16] Lily Chen, Lily Chen, Stephen Jordan, Yi-Kai Liu, Dustin Moody, Rene Peralta, Ray A Perlner, and Daniel Smith-Tone. *Report on post-quantum cryptography*, volume 12. US Department of Commerce, National Institute of Standards and Technology . . . , 2016.

[GHJ+22] Qian Guo, Clemens Hlauschek, Thomas Johansson, Norman Lahr, Alexander Nilsson, and Robin Leander Schröder. Don't reject this: Key-recovery timing attacks due to rejection-sampling in hqc and bike. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 223–263, 2022.

[GLG22a] Guillaume Goy, Antoine Loiseau, and Philippe Gaborit. A new key recovery side-channel attack on hqc with chosen ciphertext. In *International Conference on Post-Quantum Cryptography*, pages 353–371. Springer, 2022.

[GLG22b] Guillaume Goy, Antoine Loiseau, and Phlippe Gaborit. Estimating the Strength of Horizontal Correlation Attacks in the Hamming Weight Leakage Model: A Side-Channel Analysis on HQC KEM. *WCC 2022: The Twelfth International Workshop on Coding and Cryptography*, page WCC_2022_paper_48, 2022.

[GS18] Vincent Grosso and François-Xavier Standaert. Masking proofs are tight and how to exploit it in security evaluations. In *Advances in Cryptology–EUROCRYPT 2018: 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29-May 3, 2018 Proceedings, Part II 37*, pages 385–412. Springer, 2018.

[HHK17] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the fujisaki-okamoto transformation. In *Theory of Cryptography Conference*, pages 341–371. Springer, 2017.

[HHP+21] Mike Hamburg, Julius Hermelink, Robert Primas, Simona Samardjiska, Thomas Schamberger, Silvan Streit, Emanuele Strieder, and Christine van Vredendaal. Chosen ciphertext k-trace attacks on masked cca2 secure kyber. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 88–113, 2021.

[HSC+23] Senyang Huang, Rui Qi Sim, Chitchanok Chuengsatiansup, Qian Guo, and Thomas Johansson. Cache-timing attack against hqc. *Cryptology ePrint Archive*, 2023.

[HSST23] Julius Hermelink, Silvan Streit, Emanuele Strieder, and Katharina Thieme. Adapting belief propagation to counter shuffling of ntts. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 60–88, 2023.

[JH04] Jørn Justesen and Tom Høholdt. *A course in error-correcting codes*, volume 1. European Mathematical Society, 2004.

[KFL01] Frank R Kschischang, Brendan J Frey, and H-A Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on information theory*, 47(2):498–519, 2001.

[KPP20] Matthias J Kannwischer, Peter Pessl, and Robert Primas. Single-trace attacks on keccak. *Cryptology ePrint Archive*, 2020.

[Kuh55] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[LCM84] Shu Lin, Daniel J Costello, and Michael J Miller. Automatic-repeat-request error-control schemes. *IEEE Communications magazine*, 22(12):5–17, 1984.

[Mac03] David JC MacKay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[MSS13] Dominik Merli, Frederic Stumpf, and Georg Sigl. Protecting puf error correction by codeword masking. *Cryptology ePrint Archive*, 2013.

[Mun57] James Munkres. Algorithms for the assignment and transportation problems. *Journal of the society for industrial and applied mathematics*, 5(1):32–38, 1957.

[oSU23] National Institute of Standards and Technology (US). *Module-Lattice-based Key Encapsulation Mechanism Standard.* Number NIST FIPS 203. Department of Commerce, Washington, D.C., Federal Information Processing Standards Publication, 2023.

[PP19] Peter Pessl and Robert Primas. More practical single-trace attacks on the number theoretic transform. In *Progress in Cryptology–LATINCRYPT 2019: 6th International Conference on Cryptology and Information Security in Latin America, Santiago de Chile, Chile, October 2–4, 2019, Proceedings 6*, pages 130–149. Springer, 2019.

[PPM17] Robert Primas, Peter Pessl, and Stefan Mangard. Single-trace side-channel attacks on masked lattice-based encryption. In *Cryptographic Hardware and Embedded Systems–CHES 2017: 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, pages 513–533. Springer, 2017.

[PT19] Thales Bandiera Paiva and Routo Terada. A timing attack on the hqc encryption scheme. In *International Conference on Selected Areas in Cryptography*, pages 551–573. Springer, 2019.

[RPBC20] Prasanna Ravi, Romain Poussier, Shivam Bhasin, and Anupam Chattopadhyay. On configurable sca countermeasures against single trace attacks for the ntt: A performance evaluation study over kyber and dilithium on the arm cortex-m4. In *Security, Privacy, and Applied Cryptography Engineering: 10th International Conference, SPACE 2020, Kolkata, India, December 17–21, 2020, Proceedings 10*, pages 123–146. Springer, 2020.

[RRCB20] Prasanna Ravi, Sujoy Sinha Roy, Anupam Chattopadhyay, and Shivam Bhasin. Generic side-channel attacks on cca-secure lattice-based pke and kems. *IACR transactions on cryptographic hardware and embedded systems*, pages 307–335, 2020.

[SHR+22] Thomas Schamberger, Lukas Holzbaur, Julian Renner, Antonia Wachter-Zeh, and Georg Sigl. A power side-channel attack on the reed-muller reed-solomon version of the hqc cryptosystem. In *International Conference on Post-Quantum Cryptography*, pages 327–352. Springer, 2022.

[SRSWZ20] Thomas Schamberger, Julian Renner, Georg Sigl, and Antonia Wachter-Zeh. A power side-channel attack on the cca2-secure hqc kem. In *19th Smart Card Research and Advanced Application Conference (CARDIS2020)*, 2020.

[Sud00] Madhu Sudan. List decoding: Algorithms and applications. *ACM SIGACT News*, 31(1):16–27, 2000.

[UXT+22] Rei Ueno, Keita Xagawa, Yutaro Tanaka, Akira Ito, Junko Takahashi, and Naofumi Homma. Curse of re-encryption: A generic power/em analysis on post-quantum kems. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 296–322, 2022.

[VCGS14] Nicolas Veyrat-Charvillon, Benoît Gérard, and François-Xavier Standaert. Soft analytical side-channel attacks. In *Advances in Cryptology–ASIACRYPT 2014: 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, ROC, December 7-11, 2014. Proceedings, Part I 20*, pages 282–296. Springer, 2014.

[VG99] Madhu Sudan Venkatesan Guruswami. Improved decoding of reed-solomon and algebraic-geometry codes. *IEEE Transactions on Information Theory*, 45(6):1757–1767, 1999.

[WTBB+20] Guillaume Wafo-Tapa, Slim Bettaieb, Loïc Bidoux, Philippe Gaborit, and Etienne Marcatel. A practicable timing attack against hqc and its countermeasure. *Advances in Mathematics of Communications*, 2020.

A Galois Field Reduction

After the carryless multiplication performed in the HQC reference implementation, the result must be reduced to fit in the galois field. This operation is proceded with the following algorithm (See Algorithm 4). Regarding the assembly code, this algorithm is inlined inside the `gf_mul` operation, so there is no explicit call to `gf_reduce`.

Algorithm 4 Galois Field Reduction from [AMAB+]

Require: parameters: m an integer, g a generator polynomial of $\mathbb{F}_{2^m}$, Δ the distance between the primitive polynomial first two set bits
Require: x a polynomial of degre $deg(x)$
Ensure: s be the representant of x in $\mathbb{F}_{2^m}$
  steps $= \left\lceil \frac{deg(x)-(m-1)}{\Delta} \right\rceil$ ▷ Determine the number of steps
  for i from 1 to steps **do**
    $mod = x >> m$
    $x = x \oplus (1 << m) - 1 \oplus mod$
    $z_1 = 0$
    $r = g \oplus 1$
    for j from $\texttt{HW}(g) - 1$ down to 0 with step 1 **do**
      $z_2 = \texttt{number\_of\_zeros}(r)$
      $d = z_2 - z_1$
      $mod = mod << d$
      $x = x \oplus mod$
      $r = r \oplus (1 << z_2)$
      $z_1 = z_2$
  return $s := x$

B Assembly Codes

In this section, we gather assembly codes of functions we target during our side-channel analyses.

```
0   gf_mul
1   push    {lr}        Set Return Adress (link register)
2   mov     ip, r0      Set (intra procedure ) ip := r0, the first argument
3   sub     sp, #12     allocate 12 bytes of space
```

4	add	r0, sp, #4	Set r_0 to point at the allocated memory
5	movs	r3, #0	Initialized $r_3 := 0$
6	uxtb	r2, r1	Zero-extend (Padding) the lsb of r_1 into r_2
7	uxtb.w	r1, ip	Zero-extend (Padding) the lsb of ip into r_1
8	strh.w	r3, [sp, #4]	Store r_3 as a half-word at location sp +4
9	bl	<gf_carryless_mul>	▷ See Table 5
10	ldrb.w	r3, [sp, #4]	Restore r_3 from location sp +4
11	ldrb.w	r2, [sp, #5]	Load sp +5 into r_2
12	orr.w	r2, r3, r2, lsl #8	Combine r_3 and r_2 into r_2 (lsb from r_2)
			▷ − **Until 24 : `gf_reduce` of** r_2 −
13	uxtb	r3, r2	Zero-extend the lsb of r_2 into r_3 (Clear msb of r_3)
14	lsrs	r1, r2, #8	Right-shift r_2 by 8 into r_1. (Clear lsb of r_2)
15	eor.w	r3, r3, r2, lsr #8	$r_3 := (r_2 << 8) \oplus r_3$
16	eor.w	r3, r3, r1, lsl #2	$r_3 := (r_1 << 2) \oplus r_3$
17	eor.w	r3, r3, r1, lsl #3	$r_3 := (r_1 << 3) \oplus r_3$
18	eor.w	r3, r3, r1, lsl #4	$r_3 := (r_1 << 4) \oplus r_3$
19	and.w	r2, r3, #255 ; 0xff	set r_2 with the lsb of r_3
20	lsrs	r0, r3, #8	$r_0 := (r_3 >> 8)$
21	eor.w	r3, r2, r3, lsr #8	$r_3 := (r_3 << 8) \oplus r_2$
22	eor.w	r3, r3, r0, lsl #2	$r_3 := (r_0 << 2) \oplus r_3$
23	eor.w	r3, r3, r0, lsl #3	$r_3 := (r_0 << 3) \oplus r_3$
24	eor.w	r0, r3, r0, lsl #4	$r_0 := (r_0 << 4) \oplus r_3$
25	add	sp, #12	Restore the Stack, delocate the space
26	ldr.w	pc, [sp], #4	Load the return Adress

Table 4: Assembly code for Galois Field multiplication from [AMAB+] compiled with -03 optimization

0	`gf_mul_carryless`	
1	stmdb	sp!, {r4, r5, r6, r7, r8, r9, sl, fp, lr}
2	and.w	ip, r2, #127 ; 0x7f
3	sbfx	r5, r2, #7, #1
4	ubfx	r2, r1, #2, #2
5	ubfx	r3, r1, #4, #2
6	subs	r4, r2, #2
7	rsb	r8, r2, #2
8	and.w	lr, r1, #3
9	orr.w	r8, r8, r4
10	rsb	r9, r3, #2
11	subs	r4, r3, #2
12	sub.w	r7, lr, #2
13	orr.w	r9, r9, r4
14	rsb	r4, lr, #2
15	add.w	fp, r2, #4294967295 ; 0xffffffff
16	orrs	r4, r7
17	rsb	r7, r2, #1
18	orr.w	r7, r7, fp
19	mov.w	r8, r8, asr #31
20	mvn.w	r8, r8
21	asrs	r7, r7, #31
22	bic.w	r7, ip, r7
23	and.w	r8, r8, ip, lsl #1

```
24  add.w       fp, r3, #4294967295 ; 0xffffffff
25  eor.w       r8, r7, r8
26  rsb         r7, r3, #1
27  orr.w       r7, r7, fp
28  mov.w       r9, r9, asr #31
29  mvn.w       r9, r9
30  asrs        r7, r7, #31
31  and.w       r9, r9, ip, lsl #1
32  bic.w       r7, ip, r7
33  add.w       sl, lr, #4294967295 ; 0xffffffff
34  eor.w       r7, r7, r9
35  asrs        r4, r4, #31
36  rsb         r9, lr, #1
37  sub         sp, #12
38  orr.w       r9, r9, sl
39  mvns        r4, r4
40  sub.w       sl, lr, #3
41  rsb         lr, lr, #3
42  and.w       r4, r4, ip, lsl #1
43  orr.w       lr, lr, sl
44  mov.w       r9, r9, asr #31
45  eor.w       r6, ip, ip, lsl #1
46  str         r0, [sp, #4]
47  bic.w       r9, ip, r9
48  mov         r0, r4
49  mov.w       lr, lr, asr #31
50  eor.w       r9, r0, r9
51  bic.w       lr, r6, lr
52  eor.w       lr, r9, lr
53  sub.w       sl, r2, #3
54  rsb         r9, r2, #3
55  orr.w       r9, r9, sl
56  mov.w       r9, r9, asr #31
57  bic.w       r9, r6, r9
58  eor.w       r8, r8, r9
59  eor.w       r2, lr, r8, lsl #2
60  rsb         r9, r3, #3
61  sub.w       lr, r3, #3
62  orr.w       r9, r9, lr
63  mov.w       r9, r9, asr #31
64  asrs        r4, r1, #6
65  bic.w       r9, r6, r9
66  eor.w       r9, r9, r7
67  rsb         r3, r4, #2
68  subs        r7, r4, #2
69  add.w       lr, r4, #4294967295 ; 0xffffffff
70  orrs        r3, r7
71  rsb         r7, r4, #1
72  orr.w       r7, r7, lr
73  asrs        r3, r3, #31
74  asrs        r7, r7, #31
75  mvns        r3, r3
```

76	and.w	r3, r3, ip, lsl #1
77	bic.w	ip, ip, r7
78	eor.w	ip, ip, r3
79	subs	r3, r4, #3
80	rsb	r4, r4, #3
81	orrs	r4, r3
82	asrs	r4, r4, #31
83	bic.w	r4, r6, r4
84	eor.w	ip, r4, ip
85	eor.w	r2, r2, r9, lsl #4
86	mov.w	r9, r9, asr #4
87	and.w	r3, r5, r1, lsl #7
88	eor.w	r9, r9, r8, asr #6
89	eor.w	r2, r2, ip, lsl #6
90	eors	r2, r3
91	eor.w	ip, r9, ip, asr #2
92	ldr	r3, [sp, #4]
93	and.w	r1, r5, r1, lsr #1
94	eor.w	r1, ip, r1
95	strb	r2, [r3, #0]
96	strb	r1, [r3, #1]
97	add	sp, #12
98	ldmia.w	sp!, {r4, r5, r6, r7, r8, r9, sl, fp, pc}

Table 5: Assembly code for Carryless Galois Field multiplication from [AMAB$^+$] compiled with -03 optimization

C Full Shuffling Algorithm

Algorithm 5 Compute Syndromes with full shuffling strategy

Require: parameters: k, n the dimension and length of the code
Require: parity check matric $H \in \mathbb{F}_q^{(n-k,n)}$
Require: codeword $C \in \mathbb{F}_q^n$
Ensure: $s := H^T \times c$ the syndrome of c
1: Initialize s to C_1^{n-k}
2: $\mathcal{L}_1 := \{1, \cdots, n-k\}$
3: $\mathcal{L}_2 := \{2, \cdots, n\}$
4: $\mathcal{L} = \texttt{shuffle}(\mathcal{L}_1 \times \mathcal{L}_2)$
5: **for** $(i, j) \in \mathcal{L}$ **do**
6: $\quad s[i] = s[i] \oplus c[j] \times H[i, j-1]$ ▷ $\times$ is the Galois Field multiplication